Allen-Bradley

Logix5000 Controllers Import/Export

1756 ControlLogix
1756 GuardLogix
1768 CompactLogix
1769 CompactLogix
1789 SoftLogix5800
1794 FlexLogix
PwerFlex 700S with DriveLogix

Reference Manual

Rockwell Automation

Important User Information

Solid state equipment has operational characteristics differing from those of electromechanical equipment. Safety Guidelines for the Application, Installation and Maintenance of Solid State Controls (publication SGI-1.1 available from your local Rockwell Automation sales office or online at http://literature.rockwellautomation.com) describes some important differences between solid state equipment and hard-wired electromechanical devices. Because of this difference, and also because of the wide variety of uses for solid state equipment, all persons responsible for applying this equipment must satisfy themselves that each intended application of this equipment is acceptable.

In no event will Rockwell Automation, Inc. be responsible or liable for indirect or consequential damages resulting from the use or application of this equipment.

The examples and diagrams in this manual are included solely for illustrative purposes. Because of the many variables and requirements associated with any particular installation, Rockwell Automation, Inc. cannot assume responsibility or liability for actual use based on the examples and diagrams.

No patent liability is assumed by Rockwell Automation, Inc. with respect to use of information, circuits, equipment, or software described in this manual.

Reproduction of the contents of this manual, in whole or in part, without written permission of Rockwell Automation, Inc., is prohibited.

Throughout this manual, when necessary, we use notes to make you aware of safety considerations.

WARNING	Identifies information about practices or circumstances that can cause an explosion in a hazardous environment, which may lead to personal injury or death, property damage, or economic loss.
IMPORTANT	Identifies information that is critical for successful application and understanding of the product.
ATTENTION	Identifies information about practices or circumstances that can lead to: personal injury or death, property damage, or economic loss. Attentions help you identify a hazard, avoid a hazard, and recognize the consequence.
SHOCK HAZARD	Labels may be on or inside the equipment, for example, a drive or motor, to alert people that dangerous voltage may be present.
BURN HAZARD	Labels may be on or inside the equipment, for example, a drive or motor, to alert people that surfaces may reach dangerous temperatures.

Allen-Bradley, CompactLogix, ControlLogix, GuardLogix, FlexLogix, PowerFlex, DriveLogix, SoftLogix5800, Rockwell Automation, RSLinx, and RSLogix are trademarks of Rockwell Automation.

Trademarks not belonging to Rockwell Automation are property of their respective companies.

This document describes how to use version 2.7 (major revision 2, minor revision 7) of the import/export feature that is included with RSLogix 5000 programming software, version 16. Changes made to this version of the manual include:

- 1756-L61S and 1756-L62S GuardLogix safety controllers and safety relay ladder instructions (incorporated throughout the manual).
- 1756-L64 ControlLogix controller (page 28).
- Updated CONTROLLER example (page 31).
- Add-On Instructions (55...62 and page 206).
- Alarms
  - New alarm instructions: ALMA, ALMD (page 104, page 125, and page 159).
  - Digital and analog alarm tags (page 69...74).
- New instructions
  - New motion instructions: MCT, MCTP (page 107 and page 162).
  - New safety instructions: DIN, RIN, ESTOP, ENPEN, LC, FPMS, ROUT, THRS (page 105...109).
- Addition of ShareUnusedTimeSlice and InhibitAutomaticFirmwareUpdate attributes to the CONTROLLER component (page 28).
- Addition of UserDefinedVendor, UserDefinedProductType, userDefinedProductCode, UserDefinedMajor, and UserDefinedMinor attributes to the MODULE component (page 46).
- Addition of LINT data type (page 66).
- Addition of Unicast and UnicastPermitted attributes to the TAG component (page 67).
- Additional attributes and valid values for existing attributes to AXIS tags (page 75).
- Additional attributes for COORDINATE_SYSTEM tags (page 84).
- Source protected routines and Add-On-Instructions appear as encrypted data in export files. In previous releases, source protected data was not exported at all (page 62 and page 99).
- Addition of SynchronizeRedundancyDataAfterExecution attribute to the PROGRAM component (page 96).
- Additional CONFIG attributes (page 180).
- New export .TXT format for rungs and logic comments that uses tabs to separate values. This format is similar to the .CSV format that uses commas to separate values (page 16...18 and page 187...197).

  The format .CSV and .TXT formats also now include text box comments from function block and sequential function chart logic.

Notes:

Chapter 1

Import and Export Files

Introduction 13
Export a Project to a .L5K Text File. 14
Import a .L5K Text File into a Project 15
Export to a Structured File 16
Import into a Structured File. 17
Export to a .L5X XML File. 19
Import into a .L5X XML File 20
Maintaining Controller Access. 21
Structure a Complete Import/Export File 22
Conventions 23
Internal File Comments. 24
Display Style. 24
Component Descriptions 24

Chapter 2

Define a CONTROLLER Component

Introduction 27
Define a CONTROLLER. 27
Specify CONTROLLER Attributes 28
Specify CONTROLLER Attributes in a Safety Controller System. . . 29
CONTROLLER Guidelines 30
CONTROLLER Example. 31
END_CONTROLLER 34
Safety CONTROLLER Example 35

Chapter 3

Define a DATATYPE Component

Introduction 41
Define a DATATYPE. 41
Specify DATATYPE Attributes 41
Specify a DATATYPE Member 42
Specify DATATYPE Member Attributes 43
DATATYPE Guidelines 44
DATATYPE Example. 44

Chapter 4

Define a MODULE Component

Introduction 45
Define a MODULE. 45
Specify MODULE Attributes 46
Specify MODULE Attributes in a Safety Controller System. 48
Specify a MODULE Connection 49
Specify Attributes. 50
Specify Attributes in a Safety Controller System 50
MODULE Guidelines 51
MODULE Example 51
Safety Partner MODULE Example 53

Chapter 5

Define an ADD_ON_INSTRUCTION_ DEFINITION Component

Introduction 55
Define an ADD_ON_ INSTRUCTION_ DEFINITION 55
Define Routines for Add-On Instructions 56
Specify ADD_ON_ INSTRUCTION_DEFINITION Attributes 57
ADD_ON_INSTRUCTION _DEFINITION Guidelines 58
ADD_ON_INSTRUCTION _DEFINITION Example 58
Specify PARAMETERS 60
Specify PARAMETERS Attributes 60
Specify LOCAL_TAGS 61
Specify LOCAL_TAGS Attributes 61
Export Source Protected Add-On Instructions 62
Source Protected Example 63

Chapter 6

Define a TAG Component

Introduction 65
Define a Tag 65
Define a TAG Declaration for a Non-alias Tag 66
Define a TAG Declaration for an Alias Tag 67
Define an Array Specification within a TAG Declaration 67
Specify TAG Attributes 67
Specify Attributes for an ALARM_ANALOG Tag 69
Specify an ALMMSG Record for an ALARM_ANALOG Tag . . 72
Specify Parameters for an DIGITAL_ALARM Tag 72
Specify an ALMMSG Record for an ALARM_DIGITAL Tag . . 74
Specify Attributes for an AXIS_CONSUMED, AXIS_GENERIC_DRIVE, AXIS_SERVO, AXIS_SERVO_DRIVE, and AXIS_VIRTUAL Tag 75
Specify Attributes for an COORDINATE_SYSTEM Tag 84
Specify Attributes for a MESSAGE Tag 86
Specify Attributes for a MOTION_GROUP Tag 87
Specify Attributes for a SAFETY Tag 88
Define TAG Initial Values 89
Define a Comment for a TAG Component 90
TAG Guidelines 90
TAG Examples 90
Safety TAG Examples 91
ALARM_ANALOG and DIGITAL_ALARM Tag Examples 92

Chapter 7

Define a PROGRAM Component

Introduction 95
Define a PROGRAM. 95
Specify PROGRAM Attributes 96
Specify PROGRAM Attributes for EquipmentPhase Programs 97
PROGRAM Guidelines 97
PROGRAM Example 98
EquipmentPhase PROGRAM Example. 98
Safety PROGRAM Example 99
Export a Source Protected Routine. 99
Source Protected Example 100

Chapter 8

Enter Ladder Diagram Logic

Introduction 101
Enter a Ladder Logic Routine 101
Specify ROUTINE Attributes 101
Enter Rung Logic 102
Rung Guidelines. 102
Ladder ROUTINE Example. 103
Enter Branches 103
Example with a Single Branch. 104
Example with Two Simultaneous Branches 104
Enter Rung Comments 104
Enter Neutral Text for Ladder Instructions 104

Chapter 9

Enter Function Block Diagram Logic

Introduction 111
Enter a Function Block Diagram Routine 111
Specify FBD_ROUTINE Attributes 112
Enter Function Block Diagram Logic. 112
SHEET Guidelines 113
FBD_ROUTINE Example. 114
Export Function Block Logic While Editing Online 116
Example 1: Both Test Edits and Pending Edits Exist 116
Example 2: Only Pending Edits Exist 117
Enter IREFs and OREFs 117
IREF and OREF Guidelines. 118
IREF and OREF Examples 118
Enter ICONs and OCONs 119
ICON and OCON Guidelines 119
ICON and OCON Examples 120
Enter Wires and Feedback Wires 120
WIRE Guidelines. 121
WIRE Example 121

Enter Blocks 121
BLOCK Guidelines 122
Enter Text Boxes 123
TEXT_BOX Guidelines 123
TEXT_BOX Example 123
Enter Attachments 124
ATTACHMENT Guidelines 124
ATTACHMENT Example 124
Enter Parameters for Function Block Instructions 125

Chapter 10

Enter Sequential Function Chart Logic

Introduction 131
Enter a Sequential Function Chart Routine 131
Specify SFC_ROUTINE Attributes 132
SFC_ROUTINE Example 133
Export Sequential Function Chart Logic While Editing Online 139
Example 1: Both Test Edits and Pending Edits Exist 139
Example 2: Only Pending Edits Exist 140
Enter Steps 141
Enter a PRESET Block 142
Enter a LIMIT_HIGH Block 143
Enter a LIMIT_LOW Block 143
Enter an ACTION_LIST Block 143
STEP Example 145
Enter Transitions 145
Enter a CONDITION Block 146
TRANSITION Example 147
Enter Subroutine Calls 147
SBR_RET Example 148
Enter Stops 148
STOP Example 149
Enter Branches 149
Entering the LEG Block 150
BRANCH Example 150
Enter Directed Links 151
DIRECTED_LINK Guidelines 151
DIRECTED_LINK Example 151
Enter Text Boxes 152
TEXT_BOX Guidelines 152
TEXT_BOX Example 152
Enter Attachments 153
ATTACHMENT Guidelines 153
ATTACHMENT Example 153

Chapter 11

Enter Structured Text Logic

Introduction 155
Enter a Structured Text Routine 155
Specify ST_ROUTINE attributes 155
Enter Structured Text Logic 156
Structured Text ST_ROUTINE Example 157
Enter Comments 157
Export Structured Text Logic While Editing Online 158
Example 1: Both Test Edits and Pending Edits Exist 158
Example 2: Only Pending Edits Exist 158
Enter Structured Text 159

Chapter 12

Define a TASK Component

Introduction 167
Define a TASK 167
Specify TASK Attributes 168
TASK Guidelines 168
TASK Example 169
Safety TASK Example 169

Chapter 13

Define a TREND Component

Introduction 171
Define a TREND 171
Specify TREND Attributes 172
Specify a PEN Declaration 176
Specify Attributes for a PEN Declaration 176
TREND Guidelines 177
TREND Example 178

Chapter 14

Define Controller Objects

Introduction 179
Define Controller Objects 179
Specify CONFIG Attributes 180
CONFIG Examples 184

Chapter 15

Structure Tag and Comments in an Import/Export File

Introduction 187
Place Information in a .CSV or .TXT File 187
Internal File Comments 187
Specify a Tag Record 188
TAG Type Record 188
ALIAS Type Record 189
COMMENT Type Record 190
Specify a Comment Record 191
Specify an Alarm Message Record 193
Example .CSV File 194
Export All Tags and Comments 195
Example .TXT File 196
Export Program Tags and Comments 197

Chapter 16

Structure the (.L5X) Partial Import/Export File Format

Introduction 199
Identify Components in .L5X Files 201
Place Information in a Ladder Rung .L5X File 202
Define a DataType Component 202
Specify a DataType 203
Specify a Member 204
DataType Example 205
Define a Module Component 205
Define an Add-on Instruction Component 206
Add-on Instruction Example 207
Define a Tag Component 209
Tag Example 210
Define a Program Component 210
Specify a Program 210
Specify a Routine 211
Program Example 212
Example Ladder Rung .L5X File 213
Place Information in a Trend .L5X File 214
Specify a Trend 215
Trend Example 215

Appendix A

Considerations for Using Microsoft Excel to Edit a .CSV File

Introduction 217
Recommendations 217
RSLogix 5000 Data Transformations 218
Microsoft Excel Data Transformation 218

Appendix B

Import/Export Revision History

Introduction 221
Backward Compatibility 222
Import/Export Version 2.6 RSLogix 5000 Version 15 223
Import/Export Version 2.4 RSLogix 5000 Version 13 223
Import/Export Version 2.3 RSLogix 5000 Version 12 224
Import/Export Version 2.2 RSLogix 5000 Version 11 224
Import/Export Version 2.1 RSLogix 5000 Version 10 225
Changes to Support MESSAGE Tag Enhancements 225
Import/Export Version 2.0 RSLogix 5000 Version 9 227
Motion Changes to Support the SERCOS Protocol 228
MOTION_GROUP Tag Structure (Version 1.1) 229
AXIS Tag Structure (Version 1.1) 229
Import/Export Version 1.1 RSLogix 5000 Version 8 232

Notes:

Import and Export Files

Introduction

This document describes how to use version 2.7 (major revision 2, minor revision 7) of the import/export feature that is included with RSLogix 5000 programming software, version 16.

With a Logix controller, you can do a complete import/export of an entire project or you can do a partial import/export of parts of a project. The structure of the import/export file depends on whether you perform a complete or partial import/export operation. There are also considerations for complete and partial import/export operations. This chapter shows how to perform the import/export operations and describes any considerations.

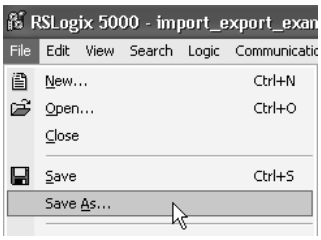
Topic	Page
Export a Project to a .L5K Text File	14
Import a .L5K Text File into a Project	15
Export to a Structured File	16
Import into a Structured File	17
Export to a .L5X XML File	19
Import into a .L5X XML File	20
Maintaining Controller Access	21
Structure a Complete Import/Export File	22

Export a Project to a .L5K Text File

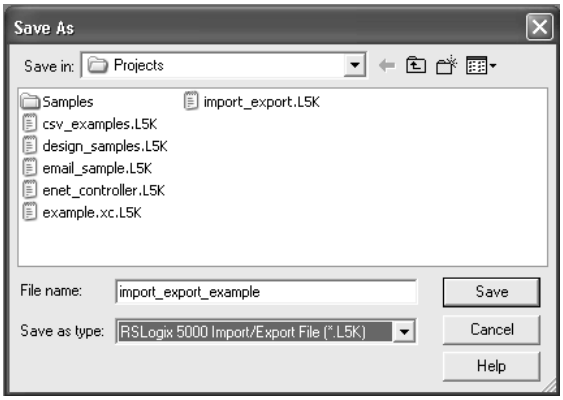
You can export a project to a text file. You can then use any text editor to modify the project.

Make sure the project you want to export is already open.

1. Select File → Save As.



2. Define the export file.



Specify the name of the text file.

Select the .L5K file format.

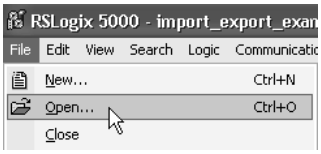
Click Save.

IMPORTANT Any unsaved edits are automatically saved when you OK the export operation.

Import a .L5K Text File into a Project

You can import controller information from a saved text file (that has a .L5K extension). This lets you use any text editor to create a project.

1. Select File → Open.



2. Select the text file.

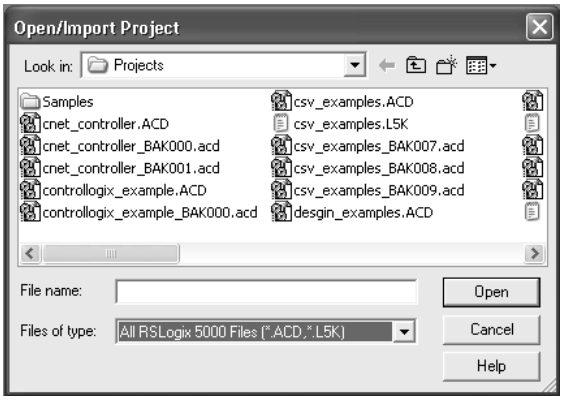
The text file must have a .L5K extension.

Select the file to import.

By default, the software points to the \RSLogix5000\Project folder. You can change the default via Tools → Options.

Specify the name for the file to import.

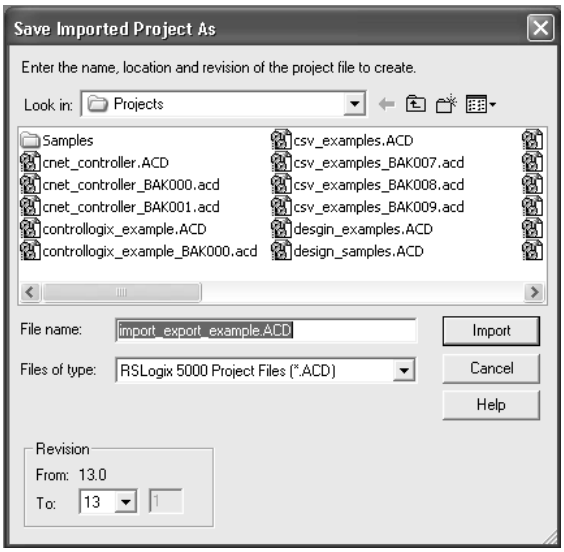
Click Open.



3. Specify the name and location of the project.

Specify the project location.

Specify the project name.



Click Import.

If you import a project that has forces, the project defaults to Forces Disabled, even if the project was exported with Forces Enabled.

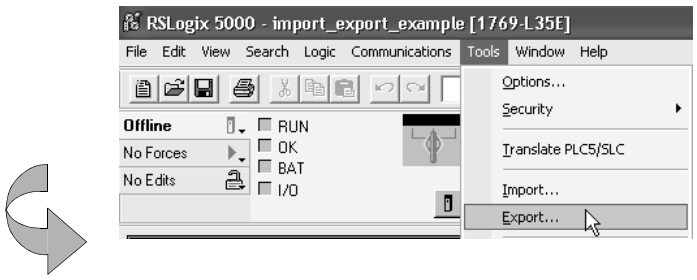
IMPORTANT When you import a .L5K file, the project changes such that you cannot go online and access a previously downloaded controller. You will have to first upload from or download to the controller. See page 21.

Export to a Structured File

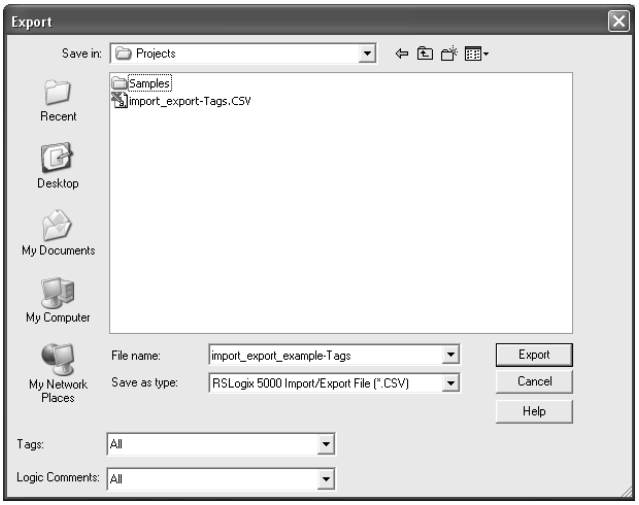
When you have a project open, you can export tags and logic comments to a structured file that separates values with commas (.CSV file) or that separates values with tabs (.TXT Unicode file). You can then use other applications (like Microsoft Excel or Notepad) to edit the tags and logic comments.

Make sure the project you want to export tags and comments from is already open.

1. Select Tools → Export.



2. Define the export file and select which tags and/or logic comments to export.



Specify the name of the export file.

Select the .CSV or .TXT file format.

Select the scope to export.

Click Export.

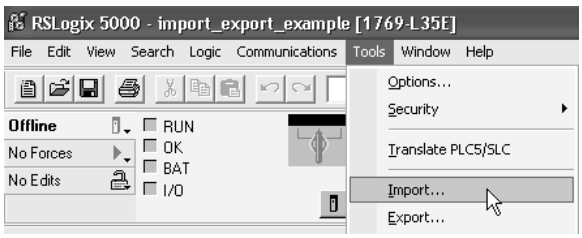
For tags and logic comments, select which content to export.

Scope	This option exports
All	All the tags (controller-scope, program-scope, equipment phase, and Add-On Instruction) or logic comments in the project
Controller and All Programs/Phases	Tags only; all controller-scope, program-scope, and equipment phase tags
Controller	Tags only; the controller-scoped tags of the project
All Programs/Phases	Logic Comments only; all program and equipment phase comments
Programs Equipment Phases Add-On Instructions	The tags or logic comments of a specific program, equipment phase, or Add-On Instruction

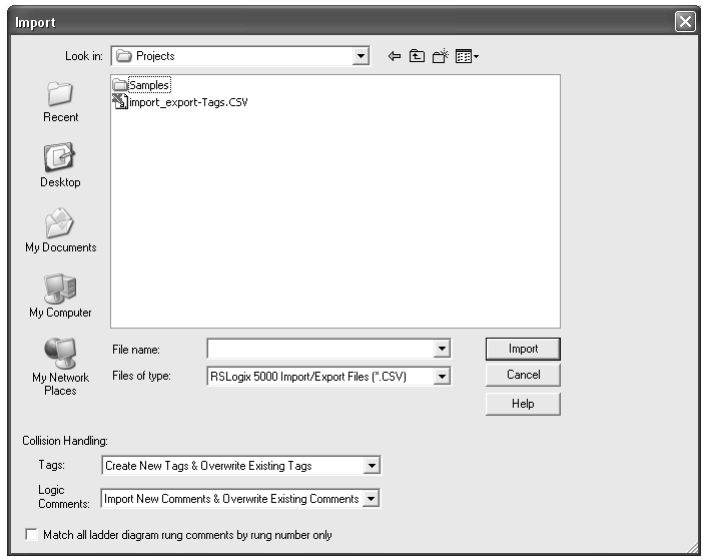
Import into a Structured File

When you are offline and have a project open, you can import tags and logic comments from a saved .CSV file or .TXT file. This lets you use other applications (like Microsoft Excel or Notepad) to create and edit tags and logic comments.

1. Select Tools → Import.



2. Select the .CSV or .TXT file to import.



Specify the name for the file to import.

Select the .CSV file or .TXT format.

Select how to handle collisions.

Select whether to match rung comments by rung number.

Click Import.

When you import tags, the possibility exists for tags in the import file to have the same name as tags already in the open project. This condition is a collision. Specify how to handle a collision.

If you want to	Select
Replace tags in the project with tags from the import file, in addition to adding any new tags from the import file.	Create New Tags & Overwrite Existing Tags (this is the default selection)
Keep tags that are in the project and discard tags in the import file, in addition to adding any new tags from the import file.	Create New Tags & Preserve Existing Tags
Replace tags in the project with tags from the import file, but do not add any new tags from the import file.	Skip New Tags & Overwrite Existing Tags

If you delete tags from an import/export file and then import the file, tags are not deleted from the controller project. You have to use the programming software to delete tags from the tag list.

When you import logic comments, the possibility exists for the comments in the import file to differ from the comments in the open project when both are matched to the same logic. Specify how to handle a collision.

If you want to	Select
Replace comments in the project with comments from the import file, in addition to adding any new comments from the import file.	Import New Comments & Overwrite Existing Comments (this is the default selection)
Keep comments that are in the project and discard comments in the import file, in addition to adding any new comments from the import file.	Import New Comments & Preserve Existing Comments
Replace comments in the project with comments from the import file, but do not add any new comments from the import file.	Skip New Comments & Overwrite Existing Comments

Also select how to match comments to logic.

If you want rung comments applied to	Then
The next rung that has the instruction, as specified in the Owning Element, as its last instruction on the rung. This is the default and recommended option. The Location element is ignored.	Leave the **Match all ladder diagram rung comments by rung only** box unchecked.
The rung number specified in the Location element. This overrides the default and recommended option. The Owning Element is ignored.	Check the **Match all ladder diagram rung comments by rung only** box.

IMPORTANT

If a .CSV file or .TXT file contains changes to tags (including aliases), when you import the file, the project changes such that you cannot go online and access a previously downloaded controller. You will have to first upload from or download to the controller.

If you only modify comments or descriptions before you import a .CSV file or .TXT file, you can go online with the controller.

Export to a .L5X XML File

If you want to re-use ladder logic from another project, export only that portion of logic to a .L5X file and import it into the required project. You can export these .L5X files.

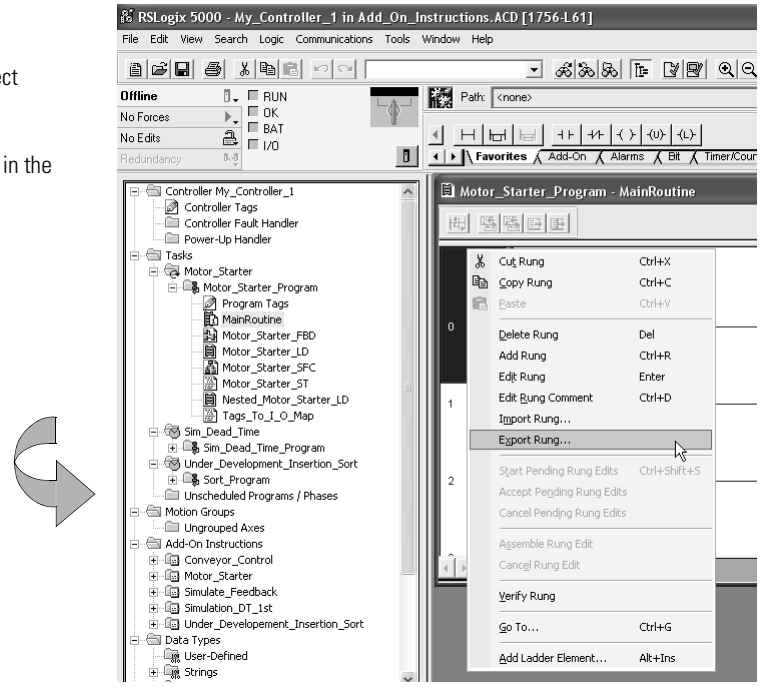
- Ladder rungs, including the referenced tags and data types
- Trends
- User defined data types
- Add-On Instructions

Make sure the project you want to export from is already open.

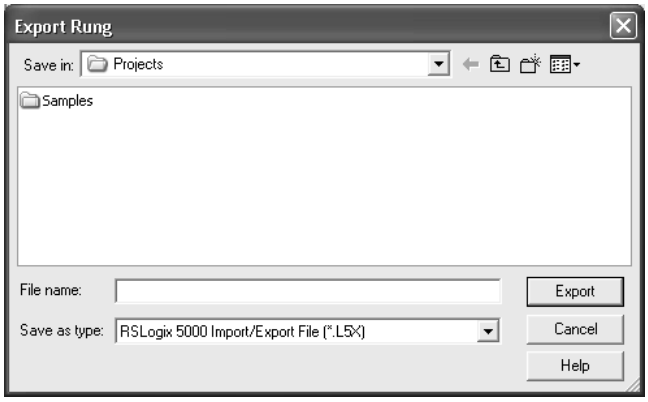
1. Select the content to export.

To export a rung, right-click on the rung and select Export Rung. You can select multiple rungs.

To export trends, right-click on the Trends folder in the Controller Organizer and select Export Trend.



2. Define the export file.



Specify the name of the export file.

Select the .L5X file format.

Click Export.

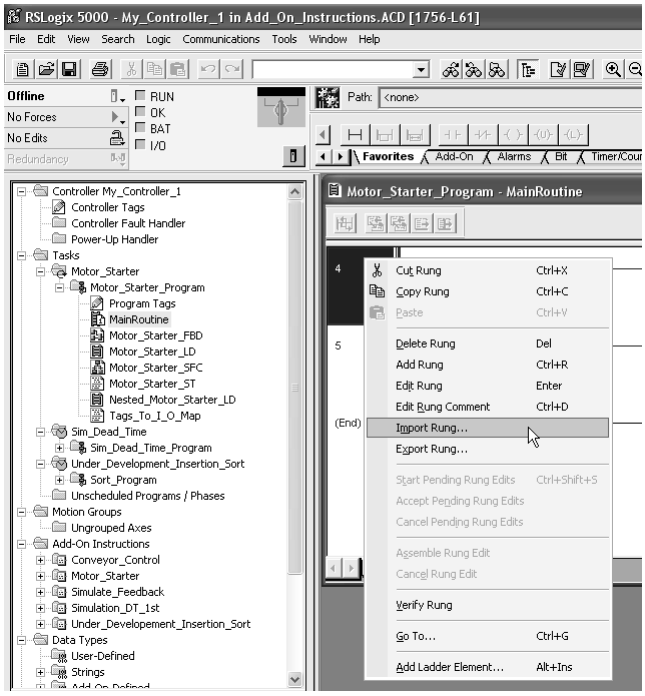
Import into a .L5X XML File

When you are offline and have a project open, you can import rungs or tags, trends, user-defined data types, and Add-On Instructions from a saved .L5X file.

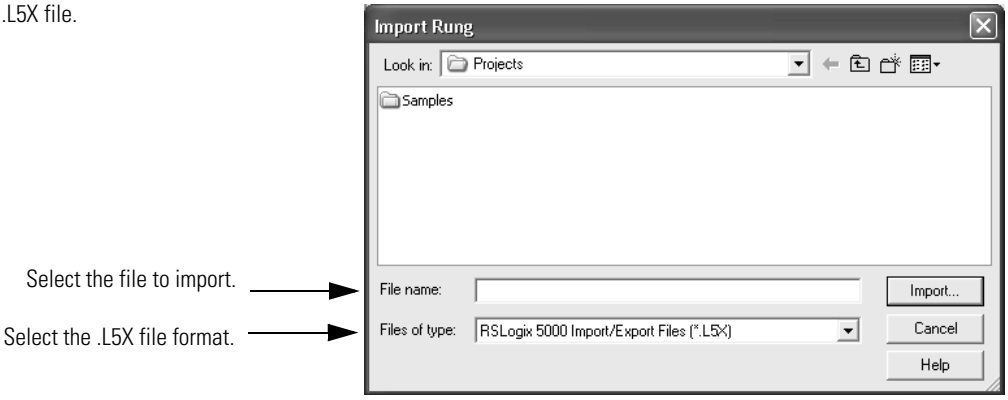
1. Select the content to import.

To import a rung, right-click where you want to insert the imported rungs and select Import Rung.

To import trends, user-defined data types, or Add-On Instructions, right-click on the correct folder in the Controller Organizer and select Import.



2. Select the .L5X file.



Click Import.

IMPORTANT When you import a .L5X file, the project changes such that you cannot go online and access a previously downloaded controller. You first upload from or download to the controller.

Maintaining Controller Access

The controller manages project status to provide RSLogix 5000 software with the information to decide whether you can go online with a controller.

Information	Description
Creation Stamp	The controller creates a creation stamp when you create a project (includes importing a project) and download the project to the controller. The creation stamp in the controller and the project file must match for RSLogix 5000 software to be able to go online with a controller. If a project is exported to a .L5K file and then imported, the resulting project .ACD file gets a new creation stamp. This means that the RSLogix 5000 software views the imported project as different from the file that was exported. The result is that you cannot use the new, imported project file to access a controller that was downloaded with the original file (before it was exported). At this point, your only options are to re-download from the imported project file or to upload the controller contents to another project .ACD file and merge with the documentation from one of the older project .ACD files.
Download Stamp	The controller creates a download stamp on each download and stores this stamp in both the project and the controller. When the creation stamp and the download stamp in the controller match those in the project file, RSLogix 5000 software can use the project to let you access the controller online. If you make changes to a project file offline, the download stamp is cleared. This can occur when you import from a .L5X file or if you import a .CSV file that creates a new tag or modifies a tag data type. When the download stamp is reset, you can either download the project to the controller or upload the contents from the controller. If you choose to upload, any changes made via import are lost. Note that description and rung comment changes in a .CSV file do not reset the download stamp so you can perform some .CSV imports and still maintain access to the controller.
Change Log	Each time you make a change on the controller online, the controllers stores details about the changes in a change log. If there are more than 1000 changes made to the project file, you must either download the project to the controller or upload the contents from the controller. If you choose to upload, any changes made via import are lost.

Given this status information, these situations cause you to not be able to go online with a controller.

Situations When You Cannot Go Online with a Controller	Possible Recovery
• More than 1000 controller edits were made. • A download of another project copy with identical stamps occurred. • Changes were made to the offline project (excluding documentation and tag value changes). • A controller nonvolatile storage load occurred and the image was downloaded from another project copy with identical stamps • A controller nonvolatile storage load occurred and the image was from an identical download, but the change log was dated earlier than the project file.	• Full download to the controller • Upload from the controller to a new project • Upload from the controller and merge with an existing project.
• The project was exported and then re-imported. In this case, the software considers it a different project and it has its own unique stamps. • A completely different project (one with different stamps) was downloaded. • A controller nonvolatile storage load occurred and the image was generated from a completely different project file (one with different stamps).	• Full download to the controller • Upload from the controller to a new project. An upload/merge of documentation is **not** possible in these cases.

Structure a Complete Import/Export File

The import/export file contains these components.

Component	Identifies
CONTROLLER	Name of the controller
DATATYPE	User-defined and I/O data structures
MODULE	Modules in the controller organizer
ADD_ON_INSTRUCTION_DEFINITION	Add-On Instructions
TAG	Controller-scope tags
PROGRAM	Program files
ROUTINE	Ladder logic routines
FBD_ROUTINE	Function block diagram routines
SFC_ROUTINE	Sequential function chart routine
ST_ROUTINE	Structured text routine
TASK	Controller tasks
TREND	Any trend configured for the controller project
CONFIG	Configuration information

All components in an import/export file follow this structure.

```
Component_Type <component_name> [Attributes]
        [body]
END_Component_Type
```

Where:

Item	Identifies
`Component_Type`	The component.
`component_name`	A specific instance of the component.
`Attributes`	Any attributes of the component. can also contain a description of the component. separate each attribute with a comma (,).
`body`	Any sub-components (children) of this component.
`END_Component_Type`	End of the component information.

Conventions

The import/export feature is based on the formats specified by the IEC 1131-3 specification.

Convention	Meaning
`< >`	Items shown in angle brackets are required.
`[ ]`	Items shown in square brackets are optional.
`user_value`	Items in italics indicate user-supplied information.
`LITERAL`	Items in all uppercase indicate a required keyword or symbol that must be entered as shown.
`"["`	Items in double quotes are required characters.

White space characters include spaces, tabs, carriage return, newline, and form feed. These characters can occur anywhere in an import/export file, except in keywords or names. If white space characters occur outside of descriptions, they are ignored.

Internal File Comments

You can enter comments to document your import files. The import process ignores these comments. You can place comments anywhere in an import/export file, except in keywords, names, component descriptions, and the value portion of attributes (before the delimiting comma or the end parenthesis).

There are two methods for entering comments.

- Start the comment with two percent (%%) characters and stop at the end of the line.
- Start the comment with a "(*" and end with a corresponding "*)". Comments can extend multiple lines.

Display Style

Tags and data types support a radix attribute that specifies how to display the associated numerical information.

Radix Display Option	Example (based on 15 decimal)
Binary (uses a 2# prefix)	2#0000_0000_0000_1111
Octal (uses a 8# prefix)	8#000_017
Decimal	15
Hex (uses a 16# prefix)	16#000F
Ascii	'$00$0F'
Exponential	1.5000000e+01
Float	15.0

Component Descriptions

Descriptions of components are optional. Unlike internal comments, descriptions are imported. Place the description within double quotes. For example:

```
TASK Task1 (Description := "Hello World", Rate := 10000,
Priority := 10 )
END_TASK
```

To enter control characters in the description, precede the character with a dollar sign ($). The following table shows how to enter the supported control characters in a description.

For this character	Enter
$	$$
‘	$’
“	$Q
10 (line feed)	$L or $l
13,10 (carriage return, line feed)	$N or $n
12 (form feed)	$P or $p
13 (carriage return)	$R or $r
9 (tab)	$T or $t
xxxx (4-digit character code that represents a hexadecimal value)	$xxxx

Notes:

Define a CONTROLLER Component

Introduction

This chapter explains the overall structure of the CONTROLLER component.

Topic	Page
Define a CONTROLLER	27
Specify CONTROLLER Attributes	28
Specify CONTROLLER Attributes in a Safety Controller System	29
CONTROLLER Guidelines	30
CONTROLLER Example	31
Safety CONTROLLER Example	35

Define a CONTROLLER

The CONTROLLER component is the overall structure of a project to be executed on one controller. It contains the configuration information and logic that you download to one controller. Preceding the CONTROLLER component, there are the header remarks (optional) and the version statement.

```
Import-Export
Version  := RSLogix 5000 16.00
Owner    := User Name, Rockwell Automation
Exported := Fri Nov 17 10:25:38 2006

IE_VER := 2.7;
```

Following the header and version statement, the CONTROLLER component follows this structure.

```
CONTROLLER <controller_name>  [Attributes]
       [<DATATYPE declaration>]
       [<MODULE declaration>]
       [<TAG declaration>]
       [<PROGRAM declaration>]
       [<TASK declaration>]
       [<CONFIG controller objects declaration>]
END_CONTROLLER
```

Where

Item	Identifies
`controller_name`	The controller name for the project.
`Attributes`	Attributes of the controller. Can also contain a description of the controller. Separate each attribute with a comma (,).
`DATATYPE`	I/O and user-defined data structures.
`MODULE`	Devices in the controller organizer.
`TAG`	Controller-scope tags.
`PROGRAM`	Organization of routines.
`TASK`	Organization of programs.
`CONFIG`	Characteristics of controller objects (status information).

Specify CONTROLLER Attributes

Specify these attributes for a CONTROLLER.

Attribute	Description
`Description`	Provide information about the controller. Specify `Description := "text"`
`ProcessorType`	Specify the type of controller (1756-L55, 1756-L60M03SE, 1756-L61, 1756-L61S, 1756-L62, 1756-L62S, 1756-L63, 1756-L64, 1768-L43, 1769-L31, 1769-L32C, 1769-L32E, 1769-L35CR, 1769-L35E, 1789-L60, 1794-L34, Emulator, PowerFlex 700S 2) Specify `ProcessorType := name`
`Major`	Specify the major revision number (1...127) of the controller. Specify `Major := number`
`TimeSlice`	Percentage of available CPU time (10...90) that is assigned to communications. Specify `TimeSlice := value`
`ShareUnusedTimeSlice`	Specify whether to share unused timeslice or not. Enter a 0 to not share; enter a 1 to share. Specify `ShareUnusedTimeSlice := number`
`PowerLossProgram`	Name of the program to be executed on reboot after a power loss. Specify `PowerLossProgram := name`
`MajorFaultProgram`	Name of the program to be executed when a major fault occurs. Specify `MajorFaultProgram := name`
`CommPath`	Specify the devices in the communication path. The communication path ends with the controller (`\Backplane\1`). This is exported only if you select manual configuration of the communications path in RSLinx software. Specify `CommPath := device\device\device...\Backplane\1`
`CommDriver`	Specify the type of communication driver. This is the name of the selected driver in RSLinx software. This is exported only if you select manual configuration of the communications driver in RSLinx software. Specify `CommDriver := text`

Attribute	Description
RedundancyEnabled	Specify whether redundancy is used or not. Enter a 0 to disable redundancy; enter a 1 to enable redundancy. Specify `RedundancyEnabled := number`
KeepTestEditsOnSwitchOver	Specify whether to keep test edits on when a switchover occurs (only in a redundant system). Enter a 0 not to keep test edits on; enter a 1 to keep test edits on. Specify `KeepTestEditsOnSwitchOver := number`
DataTablePadPercentage	Specify the percentage (0...100) of the data table to reserve. If redundancy is not enabled, enter 0. If redundancy is enabled, enter 50. Specify `DataTablePadPercentage := name`
SecurityCode	Specify whether the RSI Security Server is enabled for the controller. Enter 0 if the controller is unsecured; enter -1 if the controller is secured. Specify `SecurityCode := text`
SFCExecutionControl	Specify whether the SFC executes the current active steps before returning control (CurrentActive) or whether the SFC executes all threads until reaching a false transition (UntilFalse). Specify `SFCExecutionControl := name`
SFCRestartPosition	Specify whether the SFC restarts at the most recently executed step (MostRecent) or at the initial step (InitialStep). Specify `SFCRestartPosition := name`
SFCLastScan	Specify how the SFC manages its state on last scan. Select AutomaticReset, ProgrammaticReset, or DontScan. Specify `SFCLastScan := name`
SerialNumber	Specify the serial number of the controller. If a serial number is specified, it is imported into the project regardless of the MatchProjectToController setting. Enter a 32-bit, hexadecimal number with the 16# prefix, such as 16#0012_E2BC Specify `SerialNumber := 16#hex_string`
MatchProjectToController	Specify whether to ensure that the project matches the controller or not. Enter Yes or No. Specify `MatchProjectToController := text`
InhibitAutomaticFirmware Update	Specify whether to inhibit the automatic update of controller firmware. Enter a 0 to not inhibit; enter a 1 to inhibit. Specify `InhibitAutomaticFirmwareUpdate := number`

Specify CONTROLLER Attributes in a Safety Controller System

For GuardLogix controllers (1756-L61S, 1756-L62S), specify these attributes for the CONTROLLER component, in addition to those previously described.

Attribute	Description
SafetySignature	Specifies the safety signature control as defined in the controller properties. This value is exported only; it is ignored on import. Specify `SafetySignature := text`
SafetyLocked	Displays whether the safety controller is locked or not. This value is exported only; it is ignored on import. This value will be Yes or No. Specify `SafetyLocked := text`
SafetyLockPassword	Specifies the lock password in the controller. This value is encrypted on export. Specify `SafetyLockPassword := encrypted_characters`

Attribute	Description
`SafetyUnlockPassword`	Specifies the unlock password in the controller. This value is encrypted on export. Specify `SafetyUnlockPassword := encrypted_characters`
`SafetyTagMap`	Specify the tags in the Safety tag map. Place double quotes around the tags. Each entry must end with a comma and carriage return. For example: "StdTag1=SafeTag1, StdTag2=SafTag2" Specify `SafetyTagMap := "tag_name=tag_name"`
`ConfigureSafetyIOAlways`	Specify whether to configure safety I/O when replacing safety I/O. Enter Yes or No. Specify `ConfigureSafetyIOAlways := text`

CONTROLLER Guidelines

Keep these guidelines in mind when defining a data type.

- All declarations must be explicitly ordered as shown in the syntax above.
- The maximum number of tasks depends on the controller type.

Controller	Maximum Number of Tasks
ControlLogix	32
SoftLogix5800	32
FlexLogix	8
CompactLogix	4
DriveLogix	4

- There can be only one continuous task.
- Programs can be scheduled under only one task.
- Scheduled programs must be defined - must exist.

CONTROLLER Example

```
CONTROLLER example_controller (ProcessorType := "1756-L63",
                               Major := 16,
                               TimeSlice := 20,
                               ShareUnusedTimeSlice := 1,
                               RedundancyEnabled := 0,
                               KeepTestEditsOnSwitchOver := 0,
                               DataTablePadPercentage := 50,
                               SecurityCode := 0,
                               SFCExecutionControl :=
"CurrentActive",
                               SFCRestartPosition :=
"MostRecent",
                               SFCLastScan := "DontScan",
                               SerialNumber := 16#0000_0000,
                               MatchProjectToController := No,
                               InhibitAutomaticFirmwareUpdate
:= 0)
       MODULE Local (Parent := "Local",
                     ParentModPortId := 1,
                     CatalogNumber := "1756-L63",
                     Vendor := 1,
                     ProductType := 14,
                     ProductCode := 56,
                     Major := 16,
                     Minor := 5,
                     PortLabel := "RxBACKPLANE",
                     ChassisSize := 17,
                     Slot := 0,
                     Mode := 2#0000_0000_0000_0000,
                     CompatibleModule := 0,
                     KeyMask := 2#0000_0000_0001_1111)
       END_MODULE

       TAG
       END_TAG
```

```
        PROGRAM MainProgram (MAIN := "MainRoutine",
                             MODE := 0,
                             DisableFlag := 0)
              TAG
              END_TAG

              ROUTINE MainRoutine
              END_ROUTINE

        END_PROGRAM

        TASK MainTask (Type := CONTINUOUS,
                       Rate := 10,
                       Priority := 10,
                       Watchdog := 500,
                       DisableUpdateOutputs := No,
                       InhibitTask := No)
              MainProgram;
        END_TASK

CONFIG ASCII(XONXOFFEnable := 0,
                      DeleteMode := 0,
                      EchoMode := 0,
                      TerminationChars := 65293,
                      AppendChars := 2573,
                      BufferSize := 82) END_CONFIG

CONFIG ControllerDevice END_CONFIG

CONFIG CST(SystemTimeMasterID := 0) END_CONFIG
```

```
CONFIG DF1(DuplicateDetection := 1,
                ErrorDetection := BCC Error,
                EmbeddedResponseEnable := 0,
                DF1Mode := Pt to Pt,
                ACKTimeout := 50,
                NAKReceiveLimit := 3,
                ENQTransmitLimit := 3,
                TransmitRetries := 3,
                StationAddress := 0,
                ReplyMessageWait := 5,
                PollingMode := 1,
                MasterMessageTransmit := 0,
                NormalPollNodeFile := "<NA>",
                NormalPollGroupSize := 0,
                PriorityPollNodeFile := "<NA>",
                ActiveStationFile := "<NA>",
                SlavePollTimeout := 3000,
                EOTSuppression := 0,
                MaxStationAddress := 31,
                TokenHoldFactor := 1,
                EnableStoreFwd := 0,
                StoreFwdFile := "<NA>") END_CONFIG

CONFIG ExtendedDevice END_CONFIG

CONFIG FaultLog END_CONFIG

CONFIG FileManager END_CONFIG

CONFIG ICP END_CONFIG

CONFIG PCCC END_CONFIG

CONFIG Redundancy END_CONFIG
```

```
CONFIG SerialPort(BaudRate := 19200,
                  Parity := No Parity,
                  DataBits := 8 Bits of Data,
                  StopBits := 1 Stop Bit,
                  ComDriverId := DF1,
                  PendingComDriverId := DF1,
                  RTSOffDelay := 0,
                  RTSSendDelay := 0,
                  ControlLine := No Handshake,
                  PendingControlLine := No Handshake,
                  RemoteModeChangeFlag := 0,
                  PendingRemoteModeChangeFlag := 0,
                  ModeChangeAttentionChar := 27,
                  PendingModeChangeAttentionChar := 27,
                  SystemModeCharacter := 83,
                  PendingSystemModeCharacter := 83,
                  UserModeCharacter := 85,
                  PendingUserModeCharacter := 85,
                  DCDWaitDelay := 0) END_CONFIG

CONFIG UserMemory END_CONFIG

CONFIG WallClockTime(LocalTimeAdjustment := 0,
                           TimeZone := 0) END_CONFIG

END_CONTROLLER
CONTROLLER TestImportExport (Description := “Example",
TimeSlice := 11, MajorFaultProgram := Prg2)
       [DATATYPE declarations]
       [MODULE declarations]
       [TAG declarations]
       [PROGRAM declarations]
       [TASK declarations]
       [CONFIG controller objects declarations]

END_CONTROLLER
```

Safety CONTROLLER Example

```
CONTROLLER example_safety_controller (ProcessorType :=
"1756-L62S",
                                        Major := 16,
                                        TimeSlice := 20,
                                      ShareUnusedTimeSlice := 1,
                                        RedundancyEnabled := 0,
                                      KeepTestEditsOnSwitchOver
:= 0,
                                      DataTablePadPercentage :=
50,
                                        SecurityCode := 0,
                                        SFCExecutionControl :=
"CurrentActive",
                                        SFCRestartPosition :=
"MostRecent",
                                      SFCLastScan := "DontScan",
                                        SerialNumber :=
16#0000_0000,
                                       MatchProjectToController
:= No,
                                        SafetyLocked := No,
                                        ConfigureSafetyIOAlways
:= No,

InhibitAutomaticFirmwareUpdate := 0)
       MODULE Local (Parent := "Local",
                     ParentModPortId := 1,
                     CatalogNumber := "1756-L62S",
                     Vendor := 1,
                     ProductType := 14,
                     ProductCode := 68,
                     Major := 16,
                     Minor := 4,
                     PortLabel := "RxBACKPLANE",
                     ChassisSize := 17,
                     Slot := 0,
                     Mode := 2#0000_0000_0000_0000,
                     CompatibleModule := 0,
                     KeyMask := 2#0000_0000_0001_1111,
                     SafetyNetwork := 16#0000_31c6_0310_9358)
       END_MODULE
```

```
      MODULE example_safety_controller:Partner (Parent :=
"Local",
                                                ParentModPortId
:= 1,
                                                  CatalogNumber
:= "1756-LSP",
                                                   Vendor := 1,
                                                 ProductType :=
14,
                                                 ProductCode :=
69,
                                                   Major := 16,
                                                   Minor := 4,
                                                   PortLabel :=
"RxBACKPLANE",
                                                   Slot := 1,
                                                   Mode :=
2#0000_0000_0000_0000,
                                               CompatibleModule
:= 0,
                                                   KeyMask :=
2#0000_0000_0001_1111,
                                                  SafetyNetwork
:= 16#0000_0000_0000_0000)
      END_MODULE

      TAG
      END_TAG

      PROGRAM MainProgram (Class := Standard,
                           MAIN := "MainRoutine",
                           MODE := 0,
                           DisableFlag := 0)
            TAG
            END_TAG

            ROUTINE MainRoutine
            END_ROUTINE

      END_PROGRAM
```

```
PROGRAM SafetyProgram (Class := Safety,
                       MAIN := "MainRoutine",
                       MODE := 0,
                       DisableFlag := 0)
      TAG
      END_TAG

      ROUTINE MainRoutine
      END_ROUTINE

END_PROGRAM

TASK MainTask (Type := CONTINUOUS,
               Class := Standard,
               Rate := 10,
               Priority := 10,
               Watchdog := 500,
               DisableUpdateOutputs := No,
               InhibitTask := No)
      MainProgram;
END_TASK

TASK SafetyTask (Type := PERIODIC,
                 Class := Safety,
                 Rate := 20,
                 Priority := 10,
                 Watchdog := 20,
                 DisableUpdateOutputs := No,
                 InhibitTask := No)
      SafetyProgram;
END_TASK
```

```
CONFIG ASCII(XONXOFFEnable := 0,
                    DeleteMode := 0,
                    EchoMode := 0,
                    TerminationChars := 65293,
                    AppendChars := 2573,
                    BufferSize := 82) END_CONFIG

CONFIG ControllerDevice END_CONFIG

CONFIG CST(SystemTimeMasterID := 0) END_CONFIG

CONFIG DF1(DuplicateDetection := 1,
                  ErrorDetection := BCC Error,
                  EmbeddedResponseEnable := 0,
                  DF1Mode := Pt to Pt,
                  ACKTimeout := 50,
                  NAKReceiveLimit := 3,
                  ENQTransmitLimit := 3,
                  TransmitRetries := 3,
                  StationAddress := 0,
                  ReplyMessageWait := 5,
                  PollingMode := 1,
                  MasterMessageTransmit := 0,
                  NormalPollNodeFile := "<NA>",
                  NormalPollGroupSize := 0,
                  PriorityPollNodeFile := "<NA>",
                  ActiveStationFile := "<NA>",
                  SlavePollTimeout := 3000,
                  EOTSuppression := 0,
                  MaxStationAddress := 31,
                  TokenHoldFactor := 1,
                  EnableStoreFwd := 0,
                  StoreFwdFile := "<NA>") END_CONFIG

CONFIG ExtendedDevice END_CONFIG

CONFIG FaultLog END_CONFIG

CONFIG FileManager END_CONFIG
```

```
CONFIG ICP END_CONFIG

CONFIG PCCC END_CONFIG

CONFIG Redundancy END_CONFIG

CONFIG SafetyController END_CONFIG

CONFIG SerialPort(BaudRate := 19200,
                        Parity := No Parity,
                        DataBits := 8 Bits of Data,
                        StopBits := 1 Stop Bit,
                        ComDriverId := DF1,
                        PendingComDriverId := DF1,
                        RTSOffDelay := 0,
                        RTSSendDelay := 0,
                        ControlLine := No Handshake,
                        PendingControlLine := No Handshake,
                        RemoteModeChangeFlag := 0,
                        PendingRemoteModeChangeFlag := 0,
                        ModeChangeAttentionChar := 27,
                        PendingModeChangeAttentionChar := 27,
                        SystemModeCharacter := 83,
                        PendingSystemModeCharacter := 83,
                        UserModeCharacter := 85,
                        PendingUserModeCharacter := 85,
                        DCDWaitDelay := 0) END_CONFIG

CONFIG UserMemory END_CONFIG

CONFIG WallClockTime(LocalTimeAdjustment := 0,
                           TimeZone := 0) END_CONFIG

END_CONTROLLER
```

Notes:

Define a DATATYPE Component

Introduction

This chapter explains the overall structure of the DATATYPE component.

Topic	Page
Define a DATATYPE	41
Specify DATATYPE Attributes	41
Specify a DATATYPE Member	42
DATATYPE Guidelines	44
DATATYPE Example	44

Define a DATATYPE

A DATATYPE component follows this structure.

```
DATATYPE <DataType_name> [(Attributes)]
        [member_definition]
END_DATATYPE
```

Where:

Item	Identifies
`DataType_name`	The data structure.
`Attributes`	Attributes of the data structure. Can also contain a description of the component. Enclose in parenthesis. Separate each attribute with a comma (,).
`member_definition`	Each member of the data structure.

Specify DATATYPE Attributes

Specify these attributes for a DATATYPE.

Attribute	Description
`Description`	Provide information about the data type. Specify `Description := "text"`
`FamilyType`	Specify StringFamily for a string datatype. Specify NoFamily for all other datatypes. Specify `FamilyType := text`

Specify a DATATYPE Member

There are two kinds of data type members. A bit member is a member in which only a single bit of information is to be accessed. A non-bit member is a member that is defined as another data type (such as SINT, INT, DINT, COUNTER, etc.).

A non-bit member definition follows this structure.

```
<TypeName> <MemberName> [(Attributes)];
```

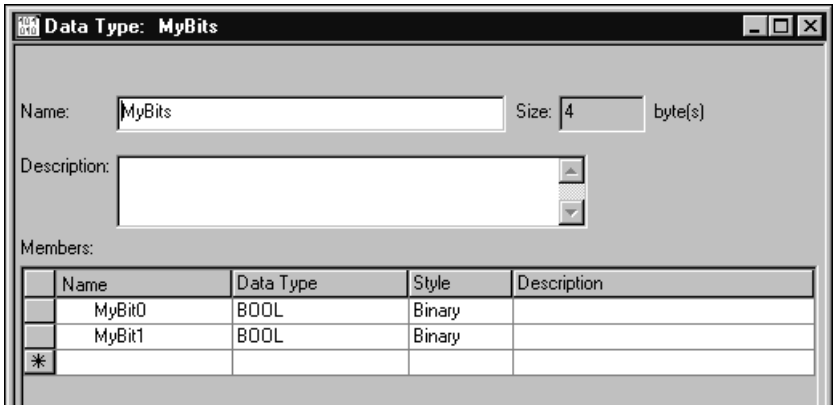
All data types are allocated in 8-bit boundaries. A single bit of storage is not allowed, so a member cannot be a BOOL data type. To access a single bit, use the BIT declaration. BIT allows access to a single bit within a host member (a non-bit member).

A bit member uses the following syntax.

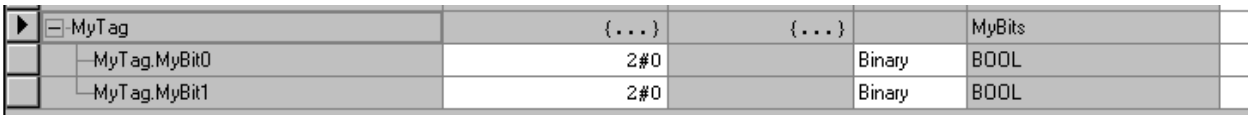
```
BIT <BitName> <HostMemberName> : <BitPosition> [(Attributes)];
```

For example, create a user-defined datatype called "MyBits" and a tag called "MyTag" of type "MyBits."

User-defined datatype MyBits.



Tag MyTag of type MyBits.



`ZZZZZZZZZZMyBits0` is the host member of MyBit0 and MyBit1. The datatype syntax for this example is:

```
DATATYPE MyBits (FamilyType := NoFamily)
       SINT ZZZZZZZZZZMyBits0 (Hidden := 1);
       BIT MyBit0 ZZZZZZZZZZMyBits0 : 0 (Radix := Binary);
       BIT MyBit1 ZZZZZZZZZZMyBits0 : 1 (Radix := Binary;
END_DATATYPE
```

The host member is normally a hidden member because only the bit references are visible when you define a tag of the datatype.

IMPORTANT There **must** be a space between the host member name and the colon and the colon and the bit position because type names can contain a colon (for example, I/O structures) and without the space we could not tell where type name actually ends.

Bit members cannot be defined before their host members. Note that BitPosition zero is the least significant bit.

Specify DATATYPE Member Attributes

Specify these attributes for a member of a DATATYPE.

Attribute	Description
`Description`	Provide information about the data type member. Specify `Description := "text"`
`Radix`	Specify decimal, hex, octal, binary, exponential, float, ASCII, or date/time. Specify `Radix := value`
`Hidden`	Make the member a hidden member of the structure. Specify `Hidden := 1`

DATATYPE Guidelines

Keep these guidelines in mind when defining a data type.

- Data types must be defined first within the controller body.
- Data types can be defined out of order. For example, if Type1 depends on Type2, Type2 can be defined first.
- Data types can be unverified. For example if Type1 depends on Type2 and Type2 is never defined, then Type1 will be accessible as an unverified type. Type2 will be typeless type. Tags of Type1 may be created but not of Type2.
- Data type members can be arrays but only one dimension is allowed.
- The following data types cannot be used in a user-defined data type: ALARM_ANALOG, ALARM_DIGITAL, AXIS types, MOTION_GROUP, and MESSAGE.
- If one user-defined data type references a second user-defined data type, the second user-defined data type must appear before the first one in the import/export file.

DATATYPE Example

```
DATATYPE MyStructure (FamilyType := NoFamily)
       DINT x;
       TIMER y[3] (Radix := Decimal);
       SINT MyFlags (Hidden :=1);
       BIT aBit0 MyFlags : 0 (Radix := Binary);
       BIT aBit1 MyFlags : 1 (Radix := Binary);
END_DATATYPE
```

Define a MODULE Component

Introduction

This chapter explains the overall structure of the MODULE component.

Topic	Page
Define a MODULE	45
Specify MODULE Attributes	46
Specify MODULE Attributes in a Safety Controller System	48
Specify a MODULE Connection	49
MODULE Guidelines	51
MODULE Example	51
Safety Partner MODULE Example	53

Define a MODULE

A MODULE component follows this structure.

```
MODULE <device_name> [(Attributes)]
       [ConfigData := <initial_value>;]
       [ExtendedProp := <text>]
       [connection_list]
END_MODULE
```

Where:

Item	Identifies
`device_name`	The module
`Attributes`	Attributes of the module. Can also contain a description of the module. Enclose in parenthesis. Separate each attribute with a comma (,).
`ConfigData`	Operating characteristics of the module.
`ExtendedProp`	Additional profile data stored in the controller. The format is XML. Currently used by the CompactBus MODULE.
`Connection`	Connection characteristics for the module.

Specify MODULE Attributes

Specify these attributes for a MODULE.

Attribute	Description
`Description`	Provide information about the module. Specify `Description := "text"`
`Parent`	If this module is a child to another module, specify the name of the parent module. The parent module must be defined before any child module. Specify `Parent := name`
`ParentModPortID`	If this module is a child to another module, specify the number of the port on the parent module that connects to this child module. The parent module must be defined before any child module. Specify `ParentModPortID := number`
`CatalogNumber`	Specify the catalog number of the module. Specify `CatalogNumber := number`
`Vendor`	Specify the vendor of the module. A number 1 indicates Allen-Bradley. Specify `Vendor := number`
`ProductType`	Specify the product type of the module. Specify `ProductType := number`
`ProductCode`	Specify the product code of the module. Specify `ProductCode := number`
`Major`	Specify the major revision number (1...127) of the module. Specify `Major := number`
`Minor`	Specify the minor revision number (1...255) of the module. Specify `Minor := number`
`UserDefinedVendor`	Specify the vendor of a non-Allen-Bradley module. Enter a number to indicate the vendor. Specify `UserDefinedVendor := number`
`UserDefinedProductType`	Specify the product type of a non-Allen-Bradley module. Specify `UserDefinedProductType := number`
`UserDefinedProductCode`	Specify the product code of a non-Allen-Bradley module. Specify `UserDefinedProductCode := number`
`UserDefinedMajor`	Specify the major revision number (1...127) of a non-Allen-Bradley module. Specify `UserDefinedMajor := number`
`UserDefinedMinor`	Specify the minor revision number (1...255) of a non-Allen-Bradley module. Specify `UserDefinedMinor := number`
`PortLabel`	Specify the port used to reach this module. The port label is either `RxBACKPLANE` for modules in a chassis or a text string for modules on a network. Specify `PortLabel := label`
`ChassisSize`	Specify the number of slots in the chassis (1...32). This applies only to the MODULE statement that defines the controller selected for the project. Specify `ChassisSize := number`
`Slot`	Specify the slot number (0...31) where the module is in the chassis. Specify `Slot := number`
`NodeAddress`	Specify the ControlNet node address (1...99) or the remote I/O rack address (0...63) of the module. Specify `NodeAddress := number`

Attribute	Description
`Group`	If the module is a remote I/O module, specify the starting group (0...7). For a block-transfer module, this is the module group number under the remote I/O adapter. Specify `Group := number`
`CommMethod`	Specify the method of connecting to the module. Specify `CommMethod := number`
`ConfigMethod`	Specify the method of configuring the module. Specify `ConfigMethod := number`
`Mode`	Select a specific mode by setting the appropriate bit. **Set** **For** 0 do not inhibit the module and a fault in the module does not cause a major fault in the controller 1 fault in the module causes a major fault in the controller 4 inhibit the module 5 both inhibit the module and a fault in the module causes a major fault in the controller Specify `Mode := number`
`CompatibleModule`	Specify whether to connect to a compatible module based on the minor revision (*value* = 1) or to an exact match of the module (*value* = 0). If you specify exact for KeyMask (below), set CompatibleModule to 2#0000_0000_0000_0000_0000_0000_0000_0000. If you specify compatible for KeyMask (below), set CompatibleModule to 2#0000_0000_0000_0000_0000_0000_1000_0000. Specify `CompatibleModule := value`
`KeyMask`	Specify whether to connect to the exact module that matches the electronic keying information (vendor, product code, product type, major revision, minor revision). No keying will connect to any module. **Specify** **To** 2#0000_0000_0000_0000 disable keying 2#0000_0000_0001_1111 require a replacement module to be compatible 2#0000_0000_0001_1111 require a replacement module to be an exact match The values for compatible module and for exact match are the same because this attribute is used in conjunction with CompatibleModule (above) to distinguish between compatible module or exact match. Specify `KeyMask := binary_string`
`PrimCxnInputSize`	Specify the size of the data associated with the primary input connection (0...500 bytes). Specify `PrimCxnInputSize := number`
`PrimCxnOutputSize`	Specify the size of the data associated with the primary output connection (0...496 bytes). Specify `PrimCxnOutputSize := number`
`SecCxnInputSize`	Specify the size of the data associated with the secondary input connection (0...500 bytes). Typically, there is one I/O connection on a module (primary connection). If there are two, the second connection is the secondary connection. Specify `SecCxnInputSize := number`
`SecCxnOutputSize`	Specify the size of the data associated with the secondary input connection (0...496 bytes). Typically, there is one I/O connection on a module (primary connection). If there are two, the second connection is the secondary connection. Specify `SecCxnOutputSize := number`

Attribute	Description
ChABaud	For a 1756-DHRIO module, specify the baud rate for channel A. Enter 57.6, 115.2, or 230.4. Specify `ChABaud := baud`
ChBBaud	For a 1756-DHRIO module, specify the baud rate for channel B. Enter 57.6, 115.2, or 230.4. Specify `ChBBaud := baud`
DtlsFileName	Specify the file name associated with a DriveExecutive project. DriveExecutive configures drives on ControlNet and EtherNet/IP networks. Specify `DtlsFileName := text`
ConfigCode	Specify the value that represents the drive rating of the drive. Select this rating on the Power tab in a DriveExecutive project for drives on ControlNet and EtherNet/IP networks. Specify `ConfigCode := text`
ControlNetSignature	This value (hexadecimal) is exported only for the purpose of doing a file compare. This value is ignored on import. The export file contains `ControlNetSignature := 16#value`
SafetyNetwork	If the module is in a safety controller system, specify the number (6-byte hexadecimal) of the safety network. Specify `SafetyNetwork := 16#value`
RSNetWorxFileName	Specify the file name of an associated RSNetWorx project file. Specify `RSNetWorxFileName := filename`

Specify MODULE Attributes in a Safety Controller System

In a safety controller system (1756-L61S, 1756-L62S), the MODULE component for the safety partner must follow the MODULE component for the primary safety controller. All of the attributes of the safety partner are determined based on those of the primary safety controller.

The MODULE component for the primary safety controller follows the structure previously described. The MODULE component for the safety partner follows this structure.

```
MODULE <device_name> [(Attributes)]
END_MODULE
```

The safety partner MODULE uses these attributes.

- Description
- Parent
- ParentModPortID
- CatalogNumber (1756-LSP)
- Vendor
- ProductType
- ProductCode
- Major
- Minor
- Mode
- PortLabel
- Slot
- CompatibleModule
- KeyMask
- SafetyNetwork

Specify a MODULE Connection

Specify these attributes for a connection.

```
CONNECTION <connection_name> [(Attributes)]
       [InputData := <value_list>;]
       [InputForceData := <value_list>;]
       [OutputData := <value_list>;]
       [OutputForceData := <value_list>;]
END_CONNECTION
```

Where:

Item	Identifies
`connection_name`	The connection.
`InputData`	Input channel data.
`InputForceData`	Forcing information for the input channel.
`OutputData`	Output channel data.
`OutputForceData`	Forcing information for the output channel.
`Attributes`	Attributes of the connection. Can also contain a description of the module. Enclose in parenthesis. Separate each attribute with a comma (,).

For details on the data in the connection list, see the user manual for the I/O module. The connection list data depends on the I/O module and the configuration for that module.

Forces appear as arrays of bytes under the InputForceData and OutputForceData attributes of the connection list. Do not modify forces in the import/export file. Use the programming software to enter and enable forces.

Specify MODULE Connection Attributes

Specify these attributes for a MODULE connection.

Attribute	Description
`Rate`	Specify the requested packet interval (RPI) rate in microseconds. Specify `Rate := microseconds`
`InputCxnPoint`	Specify the input connection point for the primary connection (0...255). Specify `InputCxnPoint := number`
`InputSize`	Specify the input size for the (0...255). Specify `InputSize := number`
`OutputCxnPoint`	Specify the output connection point for the primary connection (0...255). Specify `OutputCxnPoint := number`
`EventID`	Specify the event ID if used in conjunction with an event task. Specify `EventID := number`
`ControlNetScheduled`	This value is set by the RSNetWorx for ControlNet software when you schedule a ControlNet network. Do not modify this value.

Specify MODULE Connection Attributes in a Safety Controller System

Specify these attributes for a MODULE connection in a safety controller system, in addition to the MODULE connection attributes previously described.

Attribute	Description
`TimeoutMultiplier`	Specify the timeout multiplier (default = 2) for a safety controller system. This value determines the RPIs of time to wait for a packet before declaring a time out. This translates into the number of messages that may be lost before declaring a connection error. A Timeout Multiplier of 1 indicates that no messages may be lost; that is, there must be a packet every RPI interval. A Timeout Multiplier of 2 indicates that 1 message may be lost; that is, as long as a packet is seen in 2 times the RPI, no time-out will occur. Enter a number from 1...4, inclusive. Specify `TimeoutMultiplier := number`
`NetworkDelay Multiplier`	Specify the network delay multiplier (default = 100%) for a safety controller control system. This value lets you reduce or increase the connection reaction time limit in cases where the transport time of the message is significantly less or more than the RPI. This may be the case when the RPI of an output connection is the same as that of a lengthy task period. Enter a percentage from 10...600, inclusive. Specify `NetworkDelayMultiplier := number`
`ReactionTimeLimit`	Specify the connection reaction time limit (0...5500032) for a safety controller system. RSLogix 5000 software calculates the connection reaction time limit as a function of the RPI, timeout multiplier, and network delay multiplier. The connection reaction time limit is automatically reacalculated if any of the above values change. Specify `ReactionTimeLimit := number`

MODULE Guidelines

Keep these guidelines in mind when defining a module.

- Attributes can be in any order. They export in the order defined.
- A parent module must be defined before any definitions of its child modules.

MODULE Example

```
MODULE Local (Parent := Local,
       CatalogNumber := 1756-L1,
       Major := 1,
       PortLabel := RxBACKPLANE,
       ChassisSize := 10,
       Slot := 3,
       Mode := 2#0000_0000_0000_0000,
       CompatibleModule :=
2#0000_0000_0000_0000_0000_0000_1000_0000,
       KeyMask := 2#0000_0000_0001_1111)

END_MODULE

MODULE DHRIO_Module (Parent := Local,
       CatalogNumber := 1756-DHRIO,
       Major := 2,
       PortLabel := RxBACKPLANE,
       Slot := 8,
       CommMethod := Standard,
       ConfigMethod := ChannelA RIO ChannelB DH,
       Mode := 2#0000_0000_0000_0000,
       CompatibleModule :=
2#0000_0000_0000_0000_0000_0000_1000_0000,
       KeyMask := 2#0000_0000_0001_1111,
       ChABaud := 115.2,
       ChBBaud := 57.6)

       CONNECTION Standard (Rate := 500000,
              EventID := 0
       END_CONNECTION
END_MODULE
```

```
MODULE Diagnostic_Module_1 (Parent := Local,
       CatalogNumber := 1756-OB16D,
       Major := 1,
       PortLabel := RxBACKPLANE,
       Slot := 5,
       CommMethod := Full Diagnostics - Output Data,
       ConfigMethod := Diagnostic,
       Mode := 2#0000_0000_0000_0000,
       CompatibleModule :=
2#0000_0000_0000_0000_0000_0000_1000_0000,
       KeyMask := 2#0000_0000_0001_1111)
       ConfigData :=
[44,19,1,0,0,0,0,0,0,0,65535,65535,65535,0];

       CONNECTION Diagnostic (Rate := 5000,
              EventID := <NA>)
       END_CONNECTION

END_MODULE
MODULE input_1 (Parent := Local,
                     CatalogNumber := 1756-IA16,
                     Major := 2,
                     Minor := 1,
                     PortLabel := RxBACKPLANE,
                     Slot := 1,
                     CommMethod := 536870913,
                     ConfigMethod := 8388610,
                     Mode := 2#0000_0000_0000_0000,
                     CompatibleModule :=
2#0000_0000_0000_0000_0000_0000_1000_0000,
                     KeyMask := 2#0000_0000_0001_1111)
              ConfigData  :=
[28,16,1,0,0,0,1,9,1,9,0,0,0,0,65535,65535];
              CONNECTION StandardInput (Rate := 5000,
                                        EventID := 0)
              InputData  := [0,0];
              InputForceData :=
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,16,0,0,0,0,0,0,0,16,0];
              END_CONNECTION

       END_MODULE
```

Safety Partner MODULE Example

```
MODULE example_safety_controller:Partner (Parent := "Local",
                                          ParentModPortId := 1,
                                          CatalogNumber := "1756-LSP",
                                          Vendor := 1,
                                          ProductType := 14,
                                          ProductCode := 69,
                                          Major := 16,
                                          Minor := 4,
                                          PortLabel := "RxBACKPLANE",
                                          Slot := 1,
                                          Mode := 2#0000_0000_0000_0000,
                                          CompatibleModule := 0,
                                          KeyMask := 2#0000_0000_0001_1111,
                                          SafetyNetwork := 16#0000_0000_0000_0000)
END_MODULE
```

Notes:

Define an ADD_ON_INSTRUCTION_ DEFINITION Component

Introduction

This chapter explains the overall structure of the ADD_ON_INSTRUCTION_DEFINITION component.

Topic	Page
Define an ADD_ON_ INSTRUCTION_ DEFINITION	55
Specify ADD_ON_ INSTRUCTION_DEFINITION Attributes	57
ADD_ON_INSTRUCTION _DEFINITION Guidelines	58
ADD_ON_INSTRUCTION _DEFINITION Example	58
Specify PARAMETERS	60
Specify LOCAL_TAGS	61
Export Source Protected Add-On Instructions	62

Define an ADD_ON_ INSTRUCTION_ DEFINITION

An ADD_ON_INSTRUCTION_DEFINITION component follows this structure.

```
ADD_ON_INSTRUCTION_DEFINITION <name> [(Attributes)]
       [<PARAMETERS declaration>]
       [<LOCAL_TAGS declaration>]
       [<add_on_instruction_routines]
END_ADD_ON_INSTRUCTION_DEFINITION
```

Where:

Item	Identifies
`name`	The name of the Add-On Instruction.
`Attributes`	Attributes of the Add-On Instruction. Can also contain a description of the Add-On Instruction. Separate each attribute with a comma (,).
`PARAMETERS`	Parameters of the Add-On Instruction.
`LOCAL_TAGS`	Local tags of the Add-On Instruction.
`add_on_instruction_ routines`	Ladder logic, function block, or structured text routine. The routine name must be Logic, Prescan, Postscan, or EnableInFalse.

Define Routines for Add-On Instructions

You enter routines in an Add-On Instruction the same as you enter logic routines. The logic in a routine must all be in the same programming language. You can program the routines in ladder logic (ROUTINE), function block (FBD_ROUTINE), or structured text (ST_ROUTINE) languages. The Add-On Instruction has predefined routine names that you must use and cannot change.

Routine Name	Description
Logic	Defines the logic for the Add-On Instruction. At the minimum, every Add-On Instruction must have a Logic routine.
Prescan	Defines logic to execute during prescan.
Postscan	Defines logic to execute during postscan.
EnableInFalse	Defines logic to execute when EnableIn is false.

For example, this structure for an Add-On Instruction uses all four possible routines.

```
ADD_ON_INSTRUCTION_DEFINITION Example (attributes)

       PARAMETERS
              add_on_instruction_parameters
       END_PARAMETERS

       LOCAL_TAGS
              add_on_instruction_local_tags
       END_LOCAL_TAGS

       FBD_ROUTINE Logic (attributes)
              function_block_routine_logic
       END_FBD_ROUTINE

       ST_ROUTINE Prescan (attributes)
              structured_text_routine_logic
       END_ST_ROUTINE

       ROUTINE Postscan (attributes)
              ladder_logic_routine_logic
       END_ROUTINE

       FBD_ROUTINE EnableInFalse (attributes)
              function_block_routine_logic
       END_FBD_ROUTINE
```

If a tag in one Add-On Instruction references a second Add-On Instruction, the second Add-On Instruction must appear before the first one in the import/export file.

Specify ADD_ON_ INSTRUCTION_DEFINITION Attributes

Specify these attributes for an ADD_ON_INSTRUCTION_DEFINITION.

Attribute	Description
`Description`	Provide information about the Add-On Instruction (128 characters maximum). Specify `Description := "text"`
`Revision`	Specify the revision of the Add-On Instruction, in the form of MajorRevision.MinorRevision. Each revision number can be 1...65,535. If there is no period, the number is treated as a major revision only. Specify `Revision := "number.number"`
`RevisionExtension`	Provide additional information about the revision (40 characters maximum). Specify `RevisionExtension := "text"`
`RevisionNote`	Provide information about the revision (128 characters maximum). Specify `RevisionNote := "text"`
`Vendor`	Specify the name of the vendor (maximum 40 characters) of the Add-On Instruction. Specify `Vendor := "text"`
`ExecutePrescan`	Specify whether to execute the Prescan routine after the Logic is prescanned. Enter 1 for yes; enter 0 for no. The default is 1 if a Prescan routine exists. Specify `ExecutePrescan := number`
`ExecutePostscan`	Specify whether to execute the Postscan routine after the Logic is postscanned. Enter 1 for yes; enter 0 for no. The default is 1 if a Postscan routine exists. Specify `ExecutePostscan := number`
`ExecuteEnableInFalse`	Specify whether to execute the EnableInFalse routine when enable is false. Enter 1 for yes; enter 0 for no. The default is 1 if an EnableInFalse routine exists. Specify `ExecuteEnableInFalse := number`
`CreatedDate`	Specify the date the Add-On Instruction was created. Specify `CreatedDate := "text"`
`CreatedBy`	Specify the developer that created the Add-On Instruction. Specify `CreatedBy := "text"`
`EditedDate`	Specify the date the Add-On Instruction was last edited. Specify `EditedDate := "text"`
`EditedBy`	Specify the developer that edited the Add-On Instruction. Specify `EditedBy := "text"`
`SoftwareRevision`	Specify the revision of RSLogix 5000 software last used to edit the Add-On Instruction. The default is the currently open version of software. Specify `SoftwareRevision := number`
`AdditionalHelpText`	Specify help text specific to the Add-On Instruction. Specify `AdditionalHelpText := "text"`

ADD_ON_INSTRUCTION _DEFINITION Guidelines

Keep these guidelines in mind when defining an Add-On Instruction.

- The order of the parameters in the import file defines the order of the parameters in the Add-On Instruction.
- Local tags export in alphabetical order.
- The logic for the Add-On Instruction uses the same format and structure as the same type of logic routine in a PROGRAM component.

ADD_ON_INSTRUCTION _DEFINITION Example

```
ADD_ON_INSTRUCTION_DEFINITION Valve (Description := "Simple
valve control",
             Revision := “1.0”, RevisionExtension := “B”,
             Vendor := “RaesUDICreationsUnlimited”,
ExecutePrescan := Yes,
             ExecutePostscan := No, ExecuteEnableInFalse :=
No,
             CreatedBy := “apollo\drjones”, EditedDate :=
“2005-01-05T15:24:59.188Z”,
             EditedBy := “apollo\drjones”,
             AdditionalHelpText := “My first Add-On
Instruction – how cool!”)


       PARAMETERS
             Valve_Command : BOOL (Description := “0 - Close
valve$N1 - Open valve”,
             Radix := Decimal, Required := Yes, Visible :=
Yes, DefaultData := “1”);
             Array_Parameter : REAL[5] (Type := InOut, Radix
:= Float, Required := Yes,
             Visible := Yes); Valve_Out : DINT (Type :=
Output, Radix := Decimal,
             Required := No, Visible := Yes, DefaultData :=
“0”);
             Reset : BOOL (Description := “Used by Prescan
routine to run Reset code”,
             Type := Input, Radix := Decimal, Required := No,
Visible := No,
             DefaultData := “1”);
       END_PARAMETERS


       LOCAL_TAGS
```

```
            Valve_Type : DISCRETE_2STATE (Description :=
“The valve is a 2 state valve”,
            DefaultData :=
“[49,0.00000000e+000,0,0,0.00000000e+000,0.00000000e+000,

0.00000000e+000,0.00000000e+000,0.00000000e+000,0.00000000e+0
00]”);
      END_LOCAL_TAGS


      FBD_ROUTINE Logic (Description := "This UDI Logic
routine is nonsense but shows the
      format sufficiently. In fact, it does not even use the
InOut Parameter",
            SheetSize := "Letter (8.5x11in)",
SheetOrientation := Landscape)
            SHEET (Name := "")
            D2SD_BLOCK (ID := 0, X := 200, Y := 160, Operand
:= Valve_Type,
            VisiblePins := "ProgCommand, State0Perm,
State1Perm, FB0, FB1,
            HandFB, ProgProgReq, ProgOperReq,
ProgOverrideReq, ProgHandReq,
            Out, Device0State, Device1State, CommandStatus,
FaultAlarm,
            ModeAlarm, ProgOper, Override, Hand")
            END_D2SD_BLOCK
            IREF (ID := 1, X := 120, Y := 100, Operand :=
Valve_Command)
            END_IREF
            OREF (ID := 2, X := 460, Y := 140, Operand :=
Valve_Out)
            END_OREF
            END_SHEET
      END_FBD_ROUTINE
      ST_ROUTINE Prescan (Description := "This should run
before the Instruction does")
            '//If Reset is True - do something
            'IF (Reset) THEN
            ' //do something
            'END_IF;
            '
      END_ST_ROUTINE


END_ADD_ON_INSTRUCTION_DEFINITION
```

Specify PARAMETERS

A PARAMETERS declaration follows this structure.

```
PARAMETERS
<name> : <datatype[array_specification]> [(Attributes)];
END_PARAMETERS
```

Where:

Item	Identifies
`name`	The name of the parameter.
`datatype`	Data type of the parameter. InOut parameters can be atomic (SINT, INT, DINT, and REAL) and compound (user-defined and array) data types. In and Out parameters can be only atomic (SINT, INT, DINT, and REAL) data types.
`array`	Dimensional boundaries for an InOut parameter array.
`Attributes`	Attributes of the parameter. Can also contain a description of the parameter. Separate each attribute with a comma (,).

Specify PARAMETERS Attributes

Specify these attributes for PARAMETERS.

Attribute	Description
`Description`	Provide information about the parameter (128 characters maximum). Specify `Description := "text"`
`Usage`	Specify the type of parameter. Enter Input, Output, or InOut. Specify `Usage := text`
`Radix`	Specify decimal, hex, octal, binary, exponential, float, or ASCII. Specify `Radix := text`
`Required`	Specify whether the parameter is required. Enter 1 if the parameter is required; enter 0 if the parameter is optional. Specify `Required := number`
`Visible`	Specify whether the parameter is visible on the instruction's display. Enter 1 if the parameter is visible; enter 0 if the parameter is not visible. Specify `Visible := number`
`DefaultData`	Specify a default value for the parameter. This attribute is not available if you specify Usage as InOut. Specify `DefaultData := number`

Specify LOCAL_TAGS

A LOCAL_TAGS declaration follows this structure.

```
LOCAL_TAGS
<name> : <datatype[array_specification]> [(Attributes)];
END_LOCAL_TAGS
```

Where:

Item	Identifies
`name`	The name of the local tag.
`datatype`	Data type of the local tag. Local tags can be any data type that is supported as a member of a user-defined data type, except for Motion Group, Alarm, Axis, Coordinated, and Message types. Local tags support single-dimension arrays.
`array`	Dimensional boundaries for an array.
`Attributes`	Attributes of the local tag. Can also contain a description of the local tag. Separate each attribute with a comma (,).

Specify LOCAL_TAGS Attributes

Specify these attributes for LOCAL_TAGS.

Attribute	Description
`Description`	Provide information about the local tag (128 characters maximum). Specify `Description := "text"`
`Radix`	Specify decimal, hex, octal, binary, exponential, float, ASCII, or date/time (LINT only). Specify `Radix := text`
`DefaultData`	Specify a default value for the local tag. Specify `DefaultData := number`

Export Source Protected Add-On Instructions

If the project contains source-protected Add-On Instructions and the key is not available when you export the project, Add-On Instructions appear as ENCODED_DATA components.

```
ENCODED_DATA <EncodedType> <name> [(Attributes)]
END_ENCODED_DATA
```

Where:

Item	Identifies
`EncodedType`	The type of source-protected logic.
`name`	The name of the Add-On Instruction.
`Attributes`	Other attributes of the Add-On Instruction that are not controlled via source protection.

Source Protected Example

IMPORTANT When the source is protected, the information between the ENCODED_DATA and END_ENCODED_DATA statements is encrypted. If you modify this encrypted information in any way, you will not be able to re-import the information.

```
ENCODED_DATA (EncodedType := ADD_ON_INSTRUCTION_DEFINITION,
                    Name := "Format",
                    Description := "Configurable Data
Formatter",
                    Revision := "1.0",
                    Vendor := "ACME",
                   EditedDate := "2006-12-11T21:21:48.400Z")
xbr+GxxsZCMWfkzJFlY7U7CmecNwRm1oF+u+QYVIAy6R6eLxaznX+7SvV8IoQ
O9EBjo7LzK+N0C9OMYDT9vrJAX2DdWcPRnCnd2bjNopNxDC9XZdRVz9x2vT8r
RZSIFz69asXJRvrhVZTCBee0sO6dWpYtvqrHfLV/4hOFJFt3mB9qQTN+oWe58
9KdDw5Iz21YwT3UtlkeE8KP4b+7iwRYLmAOydRb/ygqTE01zDpZAS9AlcebPo
YUhM1gCF1AuK9eYBk0q/HI4lpDsO/VVvi7Vd7U+4/xPPRqiEjoDI25pjFZwZu
4UaGvf4jQK0NQj2wPcnu4pCbzfkNMJBPCZdfe61H4v19B7rPwxfd9Q86rl26B
u0gLeDyllG6Suoxa8OYersEhBLz6nrTYd/wybHvNmp0TLgpI1bhzWyTVttWf4
ZW8qBN6Yu796cGwaf/f5kXQ5bUni6eys/vIV/GFVQ48Z3Vg+SWC3KUawvBIhb
IhvrB1NFgcPVX7otxeGByvN5mtsVLSKw7paGvIL6n4jLC8mNJ0LuBdPtZqI5n
xSdbNSKqzhD+AqoCNtIrKoFRc1y85xGYaVV7X1HmhOnDQqokX3xPquwBiNByl
5nJ4xNg91QACSC8Xm7tQa0IPhQKfAdvQWKDVAGfH9xJarokPtx0ezxcPKut/c
m4hmBmfWxyeAoZfN01cNMtaNBdYSXe1qVv0h6tiPOhKPAyaJA7Mdg2zSRvCvi
wBDSDs8OAIykYyEHy0J0VqB1nj3eFn5e8eFa8qBvmtQhvP7VizHPN5694c8Br
/+tE7ZXIcq7UzD8r+evWQqeu2wuwbCn8L1/M7wEnEkKeYzX+dA2Un7pK3Xuc3
ABUcVOJFV3A+ZiqLGkyCJYwrGVS+OjWPu/+QauyCkkH4dd2IBcWBKgo22v1po
tIC7wrBjHbbeKMNtLlAsofwX4bKjcuWp/lXTuwu20B78GV2+W+HJu2ZsBVvVR
PLa+A+ueWZhVqAsOsMO5q1p/Z5hDGWJqEwbcDVeBYgCc+n5usIr5J1OeHh9gP
bkQz/KYWPY4XcyS2dqmWzn3rYJ/2GAbJrEJwfYZl1rV+NBeUDbvR0biyGlrcb
Ilsq/OlLqiXcAnMOGq5jW/JR8gn3uylWTuYo0wSimnSNEC82cmffZkg5RRpD5
C5ySp7go2zkVNBLuzSRTfTUN4/S9xYptSod7fGdp1sQ3IhZ3rKvz+7ifr3y9M
hwgxi7WXPOZ+0lgi/YJXLuhEv09g5lEpTv/hJ8xXeH8dNyNZ7E0O8HdNDl00X
Q

END_ENCODED_DATA
```

Notes:

Define a TAG Component

Introduction

This chapter explains the overall structure of the TAG component.

Topic	Page
Define a Tag	65
Define a TAG Declaration for a Non-alias Tag	66
Define a TAG Declaration for an Alias Tag	67
Define an Array Specification within a TAG Declaration	67
Specify TAG Attributes	67
Specify Attributes for an ALARM_ANALOG Tag	69
Specify Parameters for an DIGITAL_ALARM Tag	72
Specify Attributes for an AXIS_CONSUMED, AXIS_GENERIC_DRIVE, AXIS_SERVO, AXIS_SERVO_DRIVE, and AXIS_VIRTUAL Tag	75
Specify Attributes for an COORDINATE_SYSTEM Tag	84
Specify Attributes for a MESSAGE Tag	86
Specify Attributes for a MOTION_GROUP Tag	87
Specify Attributes for a SAFETY Tag	88
Define TAG Initial Values	89
Define a Comment for a TAG Component	90
TAG Guidelines	90
TAG Examples	90
Safety TAG Examples	91
ALARM_ANALOG and DIGITAL_ALARM Tag Examples	92

Define a Tag

Controller-scope tags are defined in one TAG component within the CONTROLLER component; program-scope tags are defined in a TAG component within a PROGRAM component within a CONTROLLER component. For example, all the tags for one program are defined in one TAG component within that PROGRAM component. A TAG component follows this structure.

```
TAG
        [tag_declarations]
END_TAG
```

Within a tag list, message and motion tags must follow all non-motion tags and axis tags must follow motion group tags.

IMPORTANT For detailed information about atomic and structure tags and their supported attributes and ranges, see the Logix5000 Controllers Common Procedures Programming Manual, publication 1756-PM001.

Define a TAG Declaration for a Non-alias Tag

A tag declaration for a non-alias tag follows this structure.

```
<tag_name> : <type[array]> [(Attributes)][:= <initial_value>] [, <tag_force_data>];
```

Where:

Item	Identifies
`tag_name`	Name of the tag.
`type`	Type of tag. Atomic types: BOOL, SINT, INT, DINT, LINT, REAL String types: STRING Predefined types: AXIS_CONSUMED, AXIS_GENERIC_DRIVE, AXIS_SERVO, AXIS_SERVO_DRIVE, AXIS_VIRTUAL, CAM, CAM_PROFILE, CONTROL, COORDINATE_SYSTEM, COUNTER, MESSAGE, MOTION_GROUP, MOTION_INSTRUCTION, OUTPUT_CAM, OUTPUT_COMPENSATION, PID, SERIAL_PORT_CONTROL, TIMER Equipment phase types: PHASE, PHASE_INSTRUCTION Safety types: CONNECTION_STATUS and unique types for each safety instruction Function block types: unique type for each function block Sequential function chart: SFC_ACTION, SFC_STEP, SFC_STOP User-defined data types.
`array`	Dimensional boundaries for array tags.
`Attributes`	Attributes of the tag. Can also contain a description of the tag. Enclose in parenthesis. Separate each attribute with a comma (,).
`initial_value`	Initial value of the tag.
`tag_forced_data`	List of forced tag data.

There **cannot** be any whitespace between the type and array definition. There **must** be a space between the tag name and the colon and another space between that same colon and the type name. This is because type names can contain a colon and without the space it would be impossible to detect where the type name actually starts.

Define a TAG Declaration for an Alias Tag

A tag declaration for an alias tag follows this structure.

```
<tag_name> OF <alias> [(Attributes)];
```

Where:

Item	Identifies
`tag_name`	Name of the alias tag.
`alias`	Name of the base tag the alias tag references. Specify `alias<specifier>` Where the *specifier* is a bit (*.bitnumber*), array element ([*element*]), or structure member (*.membername*) of the tag.
`Attributes`	Attributes of the tag. Can also contain a description of the tag. Enclose in parenthesis. Separate each attribute with a comma (,).

Define an Array Specification within a TAG Declaration

An array specification follows this structure.

```
"["<element> [,<element> [,<element>] ]"]"
```

Where:

Item	Identifies
`element`	The number of elements within the array dimension. For example [5, 10, 2].

Specify TAG Attributes

Specify these attributes for a standard TAG.

Attribute	Description
`Description`	Provide information about the tag. Specify `Description := "text"`
`Comment`	Provide information about a tag component. Specify `Comment<specifier> := "text"` Where the *specifier* is: *.bitnumber* for a bit in the tag [*element*] for an array element of the tag *.membername* for a structure member of the tag
`Class`	Specify the class of the tag. This attribute applies only to safety controller projects. Enter Standard or Safety. Specify `Class := text`
`Radix`	Specify the display style as decimal, hex, octal, binary, exponential, float, ASCII, or date/time (LINT only). Specify `Radix := value`

Attribute	Description
ProduceCount	Specify the number of consumers allowed (any positive number). Specify `ProduceCount := value`
PLCMappingFile	If this tag is mapped to a PLC controller, specify the file number (any positive number). Specify `PLCMappingFile := number`
PLC2Mapping	If this tag is mapped to a PLC-2 file, set this attribute to 1. If this tag is not mapped to a PLC-2 file, set this attribute to 0. Specify `PLC2Mapping := value`
ProgrammaticallySend EventTrigger	If the project programmatically sends an event trigger, set this attribute to 1. Otherwise, set this attribute to 0. Specify `ProgrammaticallySendEventTrigger := value`
Producer	If the controller consumes this tag, specify the name of the remote controller that produces this tag. You must also specify RemoteTag and RPI attributes. Specify `Producer:= name`
RemoteTag	If the controller consumes this tag from a controller that supports tag names, specify the name of the tag on the remote controller. You must also specify Producer and RPI attributes. Specify `RemoteTag := name`
RemoteFile	If the controller consumes this tag from a PLC-5 controller, specify the PLC-5 file number (any positive number) on the PLC-5 controller. You must also specify Producer and RPI attributes. Specify `RemoteFile := number`
RPI	If the controller consumes this tag, specify the RPI value in milliseconds (any positive number). You must also specify Producer and RemoteTag attributes. Specify `RPI := milliseconds`
Unicast	Allow connections to be unidirectional, rather than bidirectional. Enter Yes or No. Specify `Unicast := text`
UnicastPermitted	Specify when unicast connections can be received. Enter Yes or No. Specify `UnicastPermitted := text`
Usage	Specify how an Equipment Phase program uses a tag. This attribute applies only to tags that are program-scoped to an Equipment Phase program. Enter Input, Output, or Normal. Specify `Usage := text`

IMPORTANT If consume information is provided on an alias tag, the alias tag is converted to a base tag before it can consume data.

Specify Attributes for an ALARM_ANALOG Tag

An ALARM_ANALOG tag has these attributes.

Attribute	Description
EnableIn	Specify whether to enable the alarm tag. Enter 0 to disable the tag; enter 1 to enable the tag. Specify `EnableIn := BOOL`
InFault	Specify the quality of the input fault data. Enter 1 for bad quality; enter 0 for good quality. Specify `InFault := BOOL`
HHEnabled	Specify the whether the alarm monitors for a high-high limit. Enter 1 to enable; enter 0 to disable. Specify `HHEnabled := BOOL`
HEnabled	Specify the whether the alarms monitors for a high limit. Enter 1 to enable; enter 0 to disable. Specify `HEnabled := BOOL`
LEnabled	Specify the whether the alarm monitors for a low limit. Enter 1 to enable; enter 0 to disable. Specify `LEnabled := BOOL`
LLEnabled	Specify the whether the alarms monitors for a low-low limit. Enter 1 to enable; enter 0 to disable. Specify `LLEnabled := BOOL`
AckRequired	Specify whether the alarms requires acknowledgement. Enter 1 to enable; enter 0 to disable. Specify `AckRequired := BOOL`
ProgAckAll	Specify whether the program can acknowledge all alarm conditions. Enter 1 to enable; enter 0 to disable. Specify `ProgAckAll := BOOL`
OperAckAll	Specify whether an operator can acknowledge all alarm conditions. Enter 1 to enable; enter 0 to disable. Specify `OperAckAll := BOOL`
HHProgAck	Specify whether the program can acknowledge a high-high condition. Enter 1 to enable; enter 0 to disable. Specify `HHProgAck := BOOL`
HHOperAck	Specify whether an operator can acknowledge a high-high condition. Enter 1 to enable; enter 0 to disable. Specify `HHOperAck := BOOL`
HProgAck	Specify whether the program can acknowledge a high condition. Enter 1 to enable; enter 0 to disable. Specify `HProgAck := BOOL`
HOperAck	Specify whether an operator can acknowledge a high condition. Enter 1 to enable; enter 0 to disable. Specify `HOperAck := BOOL`
LProgAck	Specify whether the program can acknowledge a low condition. Enter 1 to enable; enter 0 to disable. Specify `LProgAck := BOOL`
LOperAck	Specify whether an operator can acknowledge a low condition. Enter 1 to enable; enter 0 to disable. Specify `LOperAck := BOOL`

Attribute	Description
LLProgAck	Specify whether the program can acknowledge a low-low condition. Enter 1 to enable; enter 0 to disable. Specify `LLProgAck := BOOL`
LLOperAck	Specify whether an operator can acknowledge a low-low condition. Enter 1 for enabled; enter 0 for disabled. Specify `LLOperAck := BOOL`
ROCPosProgAck	Specify whether the program can acknowledge a positive (increasing), rate-of-change condition. Enter 1 to enable; enter 0 to disable. Specify `ROCPosProgAck := BOOL`
ROCPosOperAck	Specify whether an operator can acknowledge a positive (increasing), rate-of-change condition. Enter 1 to enable; enter 0 to disable. Specify `ROCPosOperAck := BOOL`
ROCPNegProgAck	Specify whether the program can acknowledge a negative (decreasing), rate-of-change condition. Enter 1 to enable; enter 0 to disable. Specify `ROCPNegProgAck := BOOL`
ROCPNegOperAck	Specify whether an operator can acknowledge a negative (decreasing), rate-of-change condition.Enter 1 to enable; enter 0 to disable. Specify `ROCPNegOperAck := BOOL`
ProgSuppress	Specify whether the program can suppress an alarm. Enter 1 to enable; enter 0 to disable. Specify `ProgSuppress := BOOL`
OperSuppress	Specify whether an operator can suppress an alarm. Enter 1 to enable; enter 0 to disable. Specify `OperSuppress := BOOL`
ProgUnsuppress	Specify whether the program can unsuppress an alarm. Enter 1 to enable; enter 0 to disable. Specify `ProgUnsuppress := BOOL`
OperUnsuppress	Specify whether an operator can unsuppress an alarm. Enter 1 to enable; enter 0 to disable. Specify `OperUnsuppress := BOOL`
ProgDisable	Specify whether the program can disable an alarm. Enter 1 to enable; enter 0 to disable. Specify `ProgDisable := BOOL`
OperDisable	Specify whether an operator can disable an alarm. Enter 1 to enable; enter 0 to disable. Specify `OperDisable := BOOL`
ProgEnable	Specify whether the program can enable an alarm. Enter 1 to enable; enter 0 to disable. Specify `ProgEnable := BOOL`
OperEnable	Specify whether an operator can enable an alarm. Enter 1 to enable; enter 0 to disable. Specify `OperEnable := BOOL`
AlarmCountReset	Specify whether to reset the alarm count. Enter 1 to reset; enter 0 to not reset. Specify `AlarmCountReset := BOOL`
In	Specify the analog input (REAL) to the alarm. Specify `In := value`
HHLimit	Specify the high-high limit (REAL) for the alarm condition. Specify `HHLimit := value`
HHSeverity	Specify the severity (1...500) of a high-high alarm condition. Specify `HHSeverity := value`
HLimit	Specify the high limit (REAL) for the alarm condition. Specify `HLimit := value`

Attribute	Description
HSeverity	Specify the severity (1...500) of a high alarm condition. Specify `HSeverity := value`
LLimit	Specify the low limit (REAL) for the alarm condition. Specify `LLimit := value`
LSeverity	Specify the severity (1...500) of a low alarm condition. Specify `LSeverity := value`
LLLimit	Specify the low-low limit (REAL) for the alarm condition. Specify `LLLimit := value`
LLSeverity	Specify the severity (1...500) of a low-low alarm condition. Specify `LLSeverity := value`
MinDurationPRE	Specify the minimum time (DINT) an alarm condition to remain true for the alarm to be considered active. Specify `MinDurationPRE := value`
Deadband	Specify the deadband (REAL) for the high-high, high, low, and low-low levels. Specify `Deadband := value`
ROCPosLimit	Specify the positive rate-of-change limit (REAL) for the alarm condition. Specify `ROCPosLimit := value`
ROCPosSeverity	Specify the severity (1...500) of a positive rate-of-change alarm condition. Specify `ROCPosSeverity := value`
ROCNegLimit	Specify the negative rate-of-change limit (REAL) for the alarm condition. Specify `ROCNegLimit := value`
ROCNegSeverity	Specify the severity (1...500) of a negative rate-of-change alarm condition. Specify `ROCNegSeverity := value`
ROCPeriod	Specify the time period (seconds) to evaluate rate-of-change conditions. Specify `ROCPeriod := value`
AssocTag1	Specify a tag associated with the alarm. Specify `AssocTag1 := text`
AssocTag2	Specify a tag associated with the alarm. Specify `AssocTag2 := text`
AssocTag3	Specify a tag associated with the alarm. Specify `AssocTag3 := text`
AssocTag4	Specify a tag associated with the alarm. Specify `AssocTag4 := text`
HMICmd	Specify a command string for the HMI. Specify `HMICmd := text`

Specify an ALMMSG Record for an ALARM_ANALOG Tag

Each ALARM_ANALOG tag can have its own alarm message.

```
ALMMSG:<language> <scope> <tag_name> <message_text> <alarm_type>
```

Where:

Item	Identifies
`language`	Languages: EN-US (United States English), DE (Germany German), ES (Spain Spanish), FR (France French), IT (Italian), PT (Brazil Portuguese), JA (Japanese), KO (Korean), ZH (Chinese)
`scope`	Whether the alarm tag is program-scope or controller-scope.
`tag_name`	Name of the associated digital alarm tag.
`message_text`	Alarm message. Enclose the message in double quotes (" "). This is a unicode string.
`alarm_type`	Specify an analog alarm type. **Specify** **For** HH high-high alarm H high alarm L low alarm LL low-low alarm POS rate-of-change positive alarm NEG rate-of change negative alarm

Specify Parameters for an DIGITAL_ALARM Tag

A DIGITAL_ALARM tag has these attributes.

Parameter	Description
`EnableIn`	Specify whether to enable the alarm tag. Enter 0 to disable the tag; enter 1 to enable the tag. Specify `EnableIn := BOOL`
`In`	Specify the analog input to the alarm. Specify `In := BOOL`
`InFault`	Specify the quality of the input fault data. Enter 1 for bad quality; enter 0 for good quality. Specify `InFault := BOOL`
`Condition`	Specify the whether the alarm condition exists. Enter 1 for yes; enter 0 for no. Specify `Condition := BOOL`
`AckRequired`	Specify whether the alarms requires acknowledgement. Enter 1 to enable; enter 0 to disable. Specify `AckRequired := BOOL`
`Latched`	Specify the whether the alarm output is latched. Enter 1 for yes; enter 0 for no. Specify `Latched := BOOL`

Parameter	Description
ProgAck	Specify whether the program can acknowledge the alarm condition. Enter 1 to enable; enter 0 to disable. Specify `ProgAck := BOOL`
OperAck	Specify whether an operator can acknowledge the alarm condition. Enter 1 to enable; enter 0 to disable. Specify `OperAck := BOOL`
ProgReset	Specify whether the program can reset the alarm condition. Enter 1 to enable; enter 0 to disable. Specify `ProgReset := BOOL`
OperReset	Specify whether an operator can reset the alarm condition. Enter 1 to enable; enter 0 to disable. Specify `OperReset := BOOL`
ProgSuppress	Specify whether the program can suppress an alarm. Enter 1 to enable; enter 0 to disable. Specify `ProgSuppress := BOOL`
OperSuppress	Specify whether an operator can suppress an alarm. Enter 1 to enable; enter 0 to disable. Specify `OperSuppress := BOOL`
ProgUnsuppress	Specify whether the program can unsuppress an alarm. Enter 1 to enable; enter 0 to disable. Specify `ProgUnsuppress := BOOL`
OperUnsuppress	Specify whether an operator can unsuppress an alarm. Enter 1 to enable; enter 0 to disable. Specify `OperUnsuppress := BOOL`
ProgDisable	Specify whether the program can disable an alarm. Enter 1 to enable; enter 0 to disable. Specify `ProgDisable := BOOL`
OperDisable	Specify whether an operator can disable an alarm. Enter 1 to enable; enter 0 to disable. Specify `OperDisable := BOOL`
ProgEnable	Specify whether the program can enable an alarm. Enter 1 to enable; enter 0 to disable. Specify `ProgEnable := BOOL`
OperEnable	Specify whether an operator can enable an alarm. Enter 1 to enable; enter 0 to disable. Specify `OperEnable := BOOL`
AlarmCountReset	Specify whether to reset the alarm count. Enter 1 to reset; enter 0 to not reset. Specify `AlarmCountReset := BOOL`
UseProgTime	Specify how to timestamp alarm events. Enter 1 for programmatic timestamp; enter o for controller timestamp. Specify `UseProgTime := BOOL`
ProgTime	Specify the programmatic timestamp (LINT). Specify `UseProgTime := value`
Severity	Specify the severity (1...500) of the alarm condition. Specify `Severity := value`
MinDurationPRE	Specify the minimum time (DINT) an alarm condition to remain true for the alarm to be considered active. Specify `MinDurationPRE := value`
AssocTag1	Specify a tag associated with the alarm. Specify `AssocTag1 := text`
AssocTag2	Specify a tag associated with the alarm. Specify `AssocTag2 := text`

Parameter	Description
`AssocTag3`	Specify a tag associated with the alarm. Specify `AssocTag3 := text`
`AssocTag4`	Specify a tag associated with the alarm. Specify `AssocTag4 := text`
`HMICmd`	Specify a command string for the HMI. Specify `HMICmd := text`

Specify an ALMMSG Record for an ALARM_DIGITAL Tag

Each DIGITAL_ALARM tag can have its own alarm message.

```
ALMMSG:<language> <scope> <tag_name> <message_text> <alarm_type>
```

Where:

Item	Identifies
`language`	Languages: EN-US (United States English), DE (Germany German), ES (Spain Spanish), FR (France French), IT (Italian), PT (Brazil Portuguese), JA (Japanese), KO (Korean), ZH (Chinese)
`scope`	The connection.
`tag_name`	Name of the associated digital alarm tag.
`message_text`	Alarm message. Enclose the message in double quotes (" "). This is a unicode string.
`alarm_type`	Specify AM for digital alarm.

Specify Attributes for an AXIS_CONSUMED, AXIS_GENERIC_DRIVE, AXIS_SERVO, AXIS_SERVO_DRIVE, and AXIS_VIRTUAL Tag

The axis tags have these attributes.

Attribute	Description
`Description`	Provide information about the tag. Specify `Description := "text"`
`Comment`	Provide information about a tag component. Specify `Comment<specifier> := "text"` Where the *specifier* is: *.bitnumber* for a bit in the tag [*element*] for an array element of the tag *.membername* for a structure member of the tag
`MotionGroup`	Enter the name of the associated motion group, or enter <NA>. Specify `MotionGroup := text`
`MotionModule`	Enter the name of the associated motion module, or enter <NA>. Specify `MotionModule := text`
`RotationalPosResolution`	Specify the number of counts per motor revolution (1...[2^{32}-1]). Specify `RotationalPosResolution := text`
`ConversionConstant`	Specify the number of feedback counts per position unit. Enter a real number from 1.0...1.0e^{9}. Specify `ConversionConstant := value`
`OutputCamExecutionTargets`	Specify the number of output cam execution targets (any positive number). Specify `OutputCamExecutionTargets := text`
`AxisState`	Enter Axis-Ready, Direct Drive Control, Servo Control, Axis Faulted, or Axis Shutdown. Specify `AxisState := text`
`PositionUnits`	Specify user-defined engineering units (rather than feedback units). Specify `PositionUnits := text`
`AverageVelocityTimebase`	Specify the time in seconds for calculating the average velocity of the axis (any positive number). Specify `AverageVelocityTimebase := value`
`RotaryAxis`	Specify the positioning mode for an axis. Enter Rotary or Linear. Specify `RotaryAxis := text`
`PositionUnwind`	For a rotary axis, specify the distance (in feedback counts) used to perform electronic unwind (any positive number). Specify `PositionUnwind := value`
`HomeMode`	Specify the homing mode. Enter Passive, Active, or Absolute. Specify `HomeMode := text`
`HomeDirection`	For active homing sequences, except for the immediate sequence type, specify the desired homing direction. Enter Uni-directional Forward, Bi-directional Forward, Uni-directional Reverse, or Bi-directional Reverse. Specify `HomeDirection := text`
`HomeSequence`	Specify the event that will cause the home position to be set. Enter Immediate, Switch, Marker, Switch-Marker, Torque Level, or Torque Level-Marker. Specify `HomeSequence := text`

Attribute	Description
HomeConfigurationBits	Specify the home configuration bits. Enter a hexadecimal number. Specify `HomeConfigurationBits := 16#value`
HomePosition	Specify the desired absolute position, in positioning units, for the axis after the homing sequence is complete (any positive number). Specify `HomePosition := value`
HomeOffset	Specify the desired offset (any positive number) in position units the axis is to move, upon completion of the homing sequence, to reach the home position. In most cases, this value will be zero. Specify `HomeOffset := value`
HomeSpeed	Specify the speed of the jog profile used in the first leg of the homing sequence (any positive number). The homing speed should be less than the maximum speed and greater than zero. Specify `HomeSpeed := value`
HomeReturnSpeed	Specify speed of the jog profile used in the return leg(s) of an active homing sequence (any positive number). The return speed should be less than the maximum speed and greater than zero. Specify `HomeReturnSpeed := value`
MaximumSpeed	Specify the maximum speed (any positive number). Specify `MaximumSpeed := value`
MaximumAcceleration	Specify the maximum acceleration rate of the axis in position units/second (any positive number). Specify `MaximumAcceleration := value`
MaximumDeceleration	Specify the maximum deceleration rate of the axis in position units/second (any positive number). Specify `MaximumDeceleration := value`
ProgrammedStopMode	Specify how a specific axis will stop when the controller changes mode or a motion group stop (MGS) instruction is executed. Enter Fast Disable, Fast Stop, Fast Shutdown, Hard Disable, or Hard Shutdown. Specify `ProgrammedStopMode := text`
MasterInputConfigurationBits	Specify the master input configuration bits. Enter a hexadecimal number. Specify `MasterInputConfiguration := 16#value`
MasterPositionFilter Bandwidth	Specify the bandwidth in Hertz of the master position filter. Specify `MasterPositionFilterBandwidth := value`
AxisType	Specify the intended use of the axis. Enter Servo or Feedback Only. Specify `AxisType := text`
ServoLoopConfiguration	Specify the configuration of the loop. Enter Custom, Position Servo, Aux Position Servo, Dual Position Servo, Aux Command Servo, Dual Command Servo, Velocity Servo, or Torque Servo. Specify `ServoLoopConfiguration := text`
FaultConfigurationBits	Specify the fault configuration bits. Enter a hexadecimal number. Specify `FaultConfigurationBits := 16#value`

Attribute	Description
AxisInfoSelect1	Specify an axis attribute to transmit, along with the actual position data, to the controller. Enter <none>, Position Command, Position Feedback, Aux Position Feedback, Position Error, Position Int. Error, Velocity Command, Velocity Feedback, Velocity Error, Velocity Int. Error, Accel. Command, Accel. Feedback, Servo Output Level, Marker Distance, Torque Command, Torque Feedback, Positive Dynamic Torque Limit, Negative Dynamic Torque Limit, Motor Capacity, Drive Capacity, Power Capacity, Bus Regulator Capacity, Motor Electrical Angle, Torque Limit Source, DC Bus Voltage, Absolute Offset, Analog Input 1, or Analog Input 2. Specify `AxisInfoSelect1 := text`
AxisInfoSelect2	Specify a second axis attribute to transmit, along with the actual position data, to the controller. Enter <none>, Position Command, Position Feedback, Aux Position Feedback, Position Error, Position Int. Error, Velocity Command, Velocity Feedback, Velocity Error, Velocity Int. Error, Accel. Command, Accel. Feedback, Servo Output Level, Marker Distance, Torque Command, Torque Feedback, Positive Dynamic Torque Limit, Negative Dynamic Torque Limit, Motor Capacity, Drive Capacity, Power Capacity, Bus Regulator Capacity, Motor Electrical Angle, Torque Limit Source, DC Bus Voltage, Absolute Offset, Analog Input 1, or Analog Input 2. Specify `AxisInfoSelect2 := text`
LDTTYpe	Specify the LDT device type. Enter PWM, Start/Stop Rising, or Start/Stop Falling. Specify `LDTType := text`
LDTRecirculations	Only use this field if you specified PWM for LDTType. Specify the number of recirculations that the transducer is configured for so the 1756-HYD02 module knows how the LDT is configured. Specify `LDTRecirculations := value`
LDTCalibrationConstant	Specify the calibration constant (also called gradient on some LDTs). This number is engraved on each LDT by the manufacturer. It specifies the characteristics of that individual transducer. Specify `LDTCalibrationConstant := value`
LDTCalibrationConstantUnits	Specify the units of the calibration constant. Enter us/in or m/s. Specify `LDTCalibrationConstantUnits := text`
LDTScaling	Define the relationship between the unit of measurement of the transducer and the system. This is necessary for calculating the conversion constant. The LDT length is used with the number of recirculations to calculate the minimum servo update period. Specify `LDTScaling := value`
LDTScalingUnits	Specify the units of scaling. Enter us/in or m/s. Specify `LDTScalingUnits := text`
LDTLength	Specify the length of the LDT. Specify `LDTLength := value`
LDTLengthUnits	Specify the units of length. Enter us/in or m/s. Specify `LDTLengthUnits := text`
SSICodeType	Specify the encoding on the data sent from an SSI transducer. Enter Binary or Grey. Specify `SSICodeType := text`
SSIDataLength	Specify the data length (8...32 bits) of the SSI transducer. The default value is 13. Specify `SSIDataLength := text`
SSIClockFrequency	Specify the SSI clock frequency (in kHz). Valid values are 208 (default) or 650. Specify `SSIClockFrequency := value`
AbsoluteFeedbackEnable	Specify whether to enable absolute feedback. Enter 1 to enable absolute feedback. Otherwise, enter 0. Absolute feedback is always enabled for LDT. Specify `AbsoluteFeedbackEnable := value`

Attribute	Description
`AbsoluteFeedbackOffset`	Specify the absolute offset that is used to place the machine zero point at the desired location relative to the zero point of the LDT. Specify `AbsoluteFeedbackOffset := value`
`ServoFeedbackType`	Specify the type of feedback device. Enter LDT (linear displacement transducer), AQB (A quadrature B), or SSI (synchronous serial interface) Specify `ServoFeedbackType := text`
`ServoPolarityBits`	Specify the servo polarity bits. Enter a hexadecimal number. Specify `ServoPolarityBits := 16#value`
`VelocityFeedforwardGain`	Specify the velocity feedforward gain (any positive number). Specify `VelocityFeedforwardGain := value`
`AccelerationFeedforwardGain`	Specify the acceleration feedforward gain (any positive number). Specify `AccelerationFeedforwardGain := value`
`PositionProportionalGain`	Specify the position proportional gain (any positive number). Specify `ProportionalPositionGain := value`
`PositionIntegralGain`	Specify the position integral gain (any positive number). Specify `PositionIntegralGain := value`
`VelocityProportionalGain`	Specify the velocity proportional gain (any positive number). Specify `VelocityProportionalGain := value`
`VelocityIntegralGain`	Specify the velocity integral gain (any positive number). Specify `VelocityIntegralGain := value`
`VelocityScaling`	Specify the velocity scaling attribute that is used to convert the output of the servo loop into equivalent voltage to an external velocity servo drive. Specify `VelocityScaling := value`
`TorqueScaling`	Specify the torque scaling attribute that is used to convert the acceleration of the servo loop into equivalent % rated torque to the motor. Specify `TorqueScaling := value`
`OutputLPFilterBandwidth`	Specify the bandwidth in Hertz of the servo's low-pass digital output filter. Specify `OutputLPFilterBandwidth := value`
`IntegratorHoldEnable`	Enter Disabled or Enabled. Specify `IntegratorHoldEnable := value`
`PositionDifferentialGain`	Specify a position differential gain (PosD) to help predict a large overshoot ahead of time and make an attempt to correct before the overshoot actually occurs. Specify `PositionDifferentialGain := value`
`DirectionalScalingRatio`	Specify the ratio between the extend direction gain and the retract direction gain. Specify `DirectionalScalingRatio := value`
`MaximumPositiveTravel`	Specify the maximum positive position (any positive number) to be used for software overtravel checking, in position units. Specify `MaximumPositiveTravel := value`
`MaximumNegativeTravel`	Specify the maximum negative position (any positive number) to be used for software overtravel checking, in position units. Specify `MaximumNegativeTravel := value`
`PositionErrorTolerance`	Specify the how position error the servo module will tolerate (any positive number) before issuing a position error fault. Specify `PositionErrorTolerance := value`
`PositionLockTolerance`	Specify the maximum position error the servo module will accept (any positive number) in order to indicate that the position lock status bit is set. Specify `PositionLockTolerance := value`

Attribute	Description
`OutputLimit`	Specify the maximum servo output voltage of a physical axis (any positive number). Specify `OutputLimit := value`
`DirectDriveRampRate`	Specify the rate at which the analog output changes from the current value to the requested value when an MDO command is given (if ramp control is enabled). The ramp rate is specified in Volts per second. Specify `DirectDriveRampRate := value`
`OutputOffset`	Specify a fixed voltage value (-10...10V) to add to the servo output value to correct axis drift. Specify `OutputOffset := value`
`VelocityOffset`	Specify a dynamic velocity correction to the output of the position servo loop, in position units/second (any positive number). Specify `VelocityOffset := value`
`TorqueOffset`	Specify a dynamic torque command correction to the output of the velocity servo loop as a percentage of the velocity servo loop output (-100...100). Specify `TorqueOffset := value`
`FrictionCompensation`	Specify the percentage (0...100) of output level added to a positive current servo output value, or subtracted from a negative current servo output value, for the purpose of moving an axis that is stuck in place due to static friction. Specify `FrictionCompensation := value`
`FrictionCompensationWindow`	This window is defined as: command position - window attribute to command position + window attribute While the command velocity is zero and the actual position is within this window, the friction compensation (or deadband compensation, for hydraulics) is applied proportionally to the position error. While the command velocity is non-zero, the full friction compensation is applied. Specify `FrictionCompensationWindow := value`
`BacklashStabilizationWindow`	The window controls the backlash stabilization feature in the servo control loop. Mechanical backlash is a common problem in applications that utilize mechanical gearboxes. Specify `BacklashStabilizationWindow := value`
`BacklashReversalOffset`	Specify the backlash reversal error to compensate for positional inaccuracy introduced by mechanical backlash. Specify `BacklashReversalOffset := value`
`HardOvertravelFaultAction`	Specify the fault action taken when a hardware overtravel error occurs. Enter Shutdown, Disable Drive, Stop Motion, or Status Only. Specify `HardOvertravelFaultAction := text`
`SoftOvertravelFaultAction`	Specify the fault action taken when a software overtravel error occurs. Enter Shutdown, Disable Drive, Stop Motion, or Status Only. Specify `SoftOvertravelFaultAction := text`
`PositionErrorFaultAction`	Enter Shutdown, Disable Drive, Stop Motion, or Status Only. Specify `PositionErrorFaultAction := text`
`FeedbackFaultAction`	Specify the fault action to be taken when a feedback loss condition is detected. Enter Shutdown, Disable Drive, Stop Motion, or Status Only. Specify `FeedbackFaultAction := text`
`FeedbackNoiseFaultAction`	Specify the fault action to be taken when excessive feedback noise is detected. Enter Shutdown, Disable Drive, Stop Motion, or Status Only. Specify `FeedbackNoiseFaultAction := text`

Attribute	Description
`DriveFaultAction`	Specify the fault action to be taken when a drive fault condition is detected. Enter Shutdown, Disable Drive, Stop Motion, or Status Only. Specify `DriveFaultAction := text`
`TestIncrement`	Specify the amount of distance traversed by the axis when executing the output and feedback test (any positive number). Specify `TestIncrement := value`
`TuningTravelLimit`	Specify the tuning travel limit in revolutions (any positive number). Specify `TuningTravelLimit := value`
`TuningSpeed`	Specify the tuning speed in revolutions per second (any positive number). Specify `TuningSpeed := value`
`TuningTorque`	Specify the tuning torque % rated (0...300). Specify `TuningTorque := value`
`DampingFactor`	Specify the damping factor (0.5...2). Specify `DampingFactor := value`
`DriveModelTimeConstant`	Specify the drive model time constant ($1.0e^{-6f}$...1). Specify `DriveModelTimeConstant := value`
`PositionServoBandwidth`	Specify the maximum allowable value for position bandwidth (0.001F...1000), given the damping factor. This parameter is disabled if the loop configuration is set to velocity. Specify `PositionServoBandwidth := value`
`VelocityServoBandwidth`	Specify the unity gain bandwidth that is to be used to calculate the subsequent gains for a motion apply axis tuning (MAAT) instruction (0.001F...1000). Specify `VelocityServoBandwidth := value`
`TuningConfigurationBits`	Specify the tuning configuration bits. Enter a hexadecimal number. Specify `TuningConfigurationBits := 16#value`
`TorqueLimitSource`	Enter Not Limited, Negative Limit, Positive Limit, Bridge Limit, I(t) Limit, or Motor Limit. Specify `TorqueLimitSource := text`
`DriveUnit`	Specify the units of the drive. Enter us/in or m/s. Specify `DriveUnit := text`
`PositionDataScaling`	Specify the scaling method used on position values (0...255). Specify `PositionDataScaling := value`
`PositionDataScalingFactor`	Specify the scaling factor for all position data in a drive (1...65535). Specify `PositionDataScalingFactor := value`
`PositionDataScalingExp`	Specify the scaling exponent for all position data in a drive (-32768...32767). Specify `PositionDataScalingExp := value`
`VelocityDataScaling`	Specify the scaling method to use for all velocity values (0...127). Specify `VelocityDataScaling := value`
`VelocityDataScalingFactor`	Specify the scaling factor for all velocity data (1...65535). Specify `VelocityDataScalingFactor := value`
`VelocityDataScalingExp`	Specify the scaling exponent for all velocity data (-32768...32767). Specify `VelocityDataScalingExp := value`
`AccelerationDataScaling`	Specify the scaling method for all acceleration values (0...127). Specify `AccelerationDataScaling := value`
`AccelerationData ScalingFactor`	Specify the scaling factor for all acceleration data (1...65535). Specify `AccelerationDataScalingFactor := value`

Attribute	Description
AccelerationDataScalingExp	Specify the scaling exponent for all acceleration data (-32768...32767). Specify `AccelerationDataScalingExp := value`
TorqueDataScaling	Specify the scaling method for all torque values (0...127). Specify `TorqueDataScaling := value`
TorqueDataScalingFactor	Specify the scaling factor for all torque values (1...65535). Specify `TorqueDataScalingFactor := value`
TorqueDataScalingExp	Specify the scaling exponent for all torque values (-32768...32767). Specify `TorqueDataScalingExp := value`
DrivePolarity	Specify the polarity of the servo loop of the drive. Enter Custom, Positive, or Negative. Specify `DrivePolarity := text`
MotorFeedbackType	Specify the type of motor associated with the selected motor (MotorCatalogNumber). If you specify <NONE> for the motor, you must specify a feedback type. Specify `MotorFeedbackType := value`
MotorFeedbackResolution	Specify the resolution of the motor (1...2147483647). Specify `MotorFeedbackResolution := value`
AuxFeedbackType	Specify the type of auxiliary feedback device. Specify `AuxFeedbackType := value`
AuxFeedbackResolution	Specify the resolution of the auxiliary feedback device (1...2147483647). Specify `AuxFeedbackResolution := value`
MotorFeedbackUnit	Specify the units for motor feedback. Enter Rev, Inch, or Millimeter. Specify `MotorFeedbackUnit := text`
AuxFeedbackUnit	Specify the units for auxiliary feedback. Enter Rev, Inch, or Millimeter. Specify `AuxFeedbackUnit := text`
OutputNotchFilterFrequency	Specify the frequency of the drive's digital notch filer (0...10,000.0). Specify `OutputNotchFilterFrequency := value`
VelocityDroop	Specify the velocity droop (any positive number). Specify `VelocityDroop := value`
VelocityLimitBipolar	Specify the velocity limit symmetrically in both directions (any positive number). Specify `VelocityLimitBipolar := value`
AccelerationLimitBipolar	Specify the acceleration and deceleration limits for the drive (any positive number). Specify `AccelerationLimitBipolar := value`
TorqueLimitBipolar	Specify the torque limit symmetrically in both directions (0...1000.0). Specify `TorqueLimitBipolar := value`
VelocityLimitPositive	Specify the maximum allowable velocity in the positive direction (any positive number). Specify `VelocityLimitPositive := value`
VelocityLimitNegative	Specify the maximum allowable velocity in the negative direction (any positive number). Specify `VelocityLimitNegative := value`
VelocityThreshold	Specify the velocity threshold limit (any positive number). Specify `VelocityThreshold := value`
VelocityWindow	Specify the limits of the velocity window (any positive number). Specify `VelocityWindow := value`
VelocityStandstillWindow	Specify the velocity limit for the standstill window (any positive number). Specify `VelocityStandstillWindow := value`
AccelerationLimitPositive	Specify the maximum acceleration ability of the drive (any positive number). Specify `AccelerationLimitPositive := value`

Attribute	Description
AccelerationLimitNegative	Specify the maximum acceleration ability of the drive (any negative number). Specify `AccelerationLimitNegative := value`
TorqueLimitPositive	Specify the maximum torque in the positive direction (0...1000.0). Specify `TorqueLimitPositive := value`
TorqueLimitNegative	Specify the maximum torque in the negative direction (-1000.0...0). Specify `TorqueLimitNegative := value`
TorqueThreshold	Specify the torque threshold (0...1000.0). Specify `TorqueThreshold := value`
DriveThermalFaultAction	Specify the fault action to be taken when a drive thermal fault is detected. Enter Shutdown, Disable Drive, Stop Motion, or Status Only. Specify `DriveThermalFaultAction := text`
MotorThermalFaultAction	Specify the fault action to be taken when a motor thermal fault is detected. Enter Shutdown, Disable Drive, Stop Motion, or Status Only. Specify `MotorThermalFaultAction := text`
DriveEnableInputFaultAction	Specify the fault action to be taken when a drive enable input fault is detected. Enter Shutdown, Disable Drive, Stop Motion, or Status Only. Specify `MotorThermalFaultAction := text`
StoppingTorque	Specify the amount of torque available to stop the motor (0...1000). Specify `StoppingTorque := value`
StoppingTimeLimit	Specify the maximum amount of time that the drive amplifier will remain enabled while trying to stop (0...6553.5). Specify `StoppingTimeLimit := value`
BrakeEngageDelayTime	Specify the amount of time that the drive maintains torque when the servo axis is disabled and the drive decelerates to a minimum speed (0...6.5535). Specify `BrakeEngageDelayTime := value`
BrakeReleaseDelayTime	Specify amount of time that the drive ignores command values from the controller when the servo axis is enabled and the drive activates the torque (0...6.5535). Specify `BrakeReleaseDelayTime := value`
PowerSupplyID	Specify the power supply ID (any positive number). Specify `PowerSupplyID := value`
BusRegulatorID	Specify the bus regulator ID (any positive number). Specify `BusRegulatorID := value`
PWMFrequencySelect	Specify Enter High Frequency or Low Frequency. Specify `PWMFrequencySelect := text`
AmplifierCatalogNumber	Specify the catalog number of the amplifier to which this axis is connected. Specify `AmplifierCatalogNumber := text`
MotorCatalogNumber	Specify the catalog number of the motor to which this axis is connected or enter <NONE>. Specify `MotorCatalogNumber := text`
AuxFeedbackRatio	Specify the auxiliary feedback ratio (any positive number). Specify `AuxFeedbackRatio := value`
LoadInertiaRatio	Specify the load inertia ratio (any positive number). Specify `LoadInertiaRatio := value`
ContinuousTorqueLimit	Specify the maximum torque limit (0...200). Specify `ContinuousTorqueLimit := value`
ResistiveBrakeContactDelay	Specify amount of time to delay resistive brake contact. Specify `ResistiveBrakeContactDelay := value`

Attribute	Description
`ConfigurationProfile`	Specify the minimum set of attributes the drive can support. **Specify** **To** 0 Rockwell classic (identifies past systems for backward compatibility) 1 packaging (identifies packaging applications) Specify `ConfigurationProfile := value`
`RegistrationInputs`	Specify the number of drive-resident (probe) inputs. You can have as many as two registration inputs per axis. Specify `RegistrationInputs := value`
`RegistrationInputs`	Specify the number of drive-resident (probe) inputs. You can have as many as two registration inputs per axis. Specify `RegistrationInputs := value`
`MaximumAccelerationJerk`	Specify the value motion instructions use to determine the acceleration jerk to apply to the axis when acceleration jerk is specified as a percent of the maximum. Specify `MaximumAccelerationJerk := value`
`MaximumDecelerationJerk`	Specify the value motion instructions use to determine the deceleration jerk to apply to the axis when deceleration jerk is specified as a percent of the maximum. This value is only used by a S-curve profile. Specify `MaximumDecelerationJerk := value`
`DynamicsConfigurationBits`	Specify the S-curve profile. **Specify** **To** 0 reduce S-curve stop delay 1 prevent S-curve velocity reversals Specify `DynamicsConfigurationBits := value`
`PhaseLossFaultAction`	Specify how the axis responds to a drive fault. The default is 1 (disable drive). **Specify** **To** 0 shutdown 1 disable drive 2 stop command 3 status only Specify `PhaseLossFaultAction := value`
`HomeTorqueLevel`	Specify the torque limit when using one of the torque homing modes. Enter the percent (0...TorqueLimitPositive) of continuous torque. The default is 0%. Specify `HomeTorqueLevel := value`
`InputPowerPhase`	Specify the power phase operation of a Kinetix 2000 drive. Enter 0 for three-phase power; enter 1 for single-phase power. Specify `InputPowerPhase := number`

Specify Attributes for an COORDINATE_SYSTEM Tag

The COORDINATE_SYSTEM tag has these attributes.

Attribute	Description
`Description`	Provide information about the tag. Specify `Description := "text"`
`Comment`	Provide information about a tag component. Specify `Comment<specifier> := "text"` Where the *specifier* is: *.bitnumber* for a bit in the tag [*element*] for an array element of the tag *.membername* for a structure member of the tag
`MotionGroupInstance`	Enter the name of the associated motion group, or enter <NA>. Specify `MotionGroupInstance := text`
`SystemType`	Specify the coordinate system type. Currently, only Cartesian is available. Specify `SystemType := Cartesian`
`Dimension`	Specify the number of axes that this coordinated system supports. Enter 1, 2, or 3. Specify `Dimension := value`
`Axes`	Specify the name of the axes in this coordinated system. Specify `Axes := value`
`CoordinationMode`	Specify coordination mode. Currently, only Primary is available. Specify `CoordinationMode := Primary`
`CoordinationUnits`	Specify units to be used for measuring and calculating motion related values such as position, velocity, etc. Enter units that are relevant to your application. Specify `CoordinationUnits := text`
`ConversionRatioNumerator`	The conversion ratio defines the relationship of axis position units to coordination units for each axis. Enter the numerator as a float or an integer. Specify `ConversionRatioNumerator := value`
`ConversionRatioDenominator`	The conversion ratio defines the relationship of axis position units to coordination units for each axis. Enter the denominator as an integer. Specify `ConversionRatioDenominator := value`
`CoordinateSystemAutoTag Update`	Specify whether or not the actual position values of the current coordinated system are automatically updated during operation. To enable auto tag update, enter 1. Otherwise, enter 0. Specify `CoordinateSystemAutoTagUpdate := text`
`MaximumSpeed`	Specify the maximum speed to be used by the coordinated motion instructions in calculating vector speed when speed is expressed as a percent of maximum. Specify `MaximumSpeed := value`
`MaximumAcceleration`	Specify the value for maximum acceleration to be used by the coordinated motion instructions to determine the acceleration rate to apply to the coordinate system vector when acceleration is expressed as a percent of maximum. Specify `MaximumAccelaertion := value`
`MaximumDeceleration`	Specify the value for maximum deceleration to be used by the coordinated motion instructions to determine the deceleration rate to apply to the coordinate system vector when deceleration is expressed as a percent of maximum. Specify `MaximumAccelaertion := value`

Attribute	Description
ActualPositionTolerance	Specify the value in coordination units, for actual position to be used by coordinated motion instructions when they have a termination type of actual tolerance. Specify `ActualPositionTolerance := value`
CommandPositionTolerance	Specify the value in coordination units, for command position to be used by coordinated motion instructions when they have a termination type of command tolerance. Specify `CommandPositionTolerance := value`
TransformDimension	Specify the transform dimension. Specify `TransformDimension := value`
JointRatioNumerator	Specify numerator of the joint ratio. Specify `JointRatioNumerator := value`
JointRatioDenominator	Specify denominator of the joint ratio. Specify `JointRatioDenominator := value`
LinkLength1	Specify the length of Robotic Arm 1. Specify `LinkLength1 := value`
LinkLength2	Specify the length of Robotic Arm 2. Specify `LinkLength2 := value`
ZeroAngleOffset1	Specify the rotational angular offset of joint axes 1 in degrees. This is used to shift the 0 degree position of the joint. Specify `ZeroAngleOffset1 := value`
ZeroAngleOffset2	Specify the rotational angular offset of joint axes 2 in degrees. This is used to shift the 0 degree position of the joint. Specify `ZeroAngleOffset2 := value`
ZeroAngleOffset3	Specify the rotational angular offset of joint axes 3 in degrees. This is used to shift the 0 degree position of the joint. Specify `ZeroAngleOffset3 := value`
BaseOffset1	Specify the difference for the first axis between the origin of the robot at the first joint of the robotic arm (as determined by RSLogix 5000 Kinematics internal equations) and the origin defined by the robot manufacturer. Specify `BaseOffset1 := value`
BaseOffset2	Specify the difference for the second axis between the origin of the robot at the first joint of the robotic arm (as determined by RSLogix 5000 Kinematics internal equations) and the origin defined by the robot manufacturer. Specify `BaseOffset2 := value`
BaseOffset3	Specify the difference for the third axis between the origin of the robot at the first joint of the robotic arm (as determined by RSLogix 5000 Kinematics internal equations) and the origin defined by the robot manufacturer. Specify `BaseOffset3 := value`
EndEffectorOffset1	Specify the end effector offset value, which is the distance between the end of the robot arm L2 and the end of the end effector in the x1 dimension. Specify `EndEffectorOffset1 := value`
EndEffectorOffset2	Specify the end effector offset value, which is the distance between the end of the robot arm L2 and the end of the end effector in the x2 dimension. Specify `EndEffectorOffset2 := value`
EndEffectorOffset3	Specify the end effector offset value, which is the distance between the end of the robot arm L2 and the end of the end effector in the x3 dimension. Specify `EndEffectorOffset3 := value`

Specify Attributes for a MESSAGE Tag

A MESSAGE tag has these attributes.

Attribute	Description
`Description`	Provide information about the tag. Specify `Description := "text"`
`Comment`	Provide information about a tag component. Specify `Comment<specifier> := "text"` Where the *specifier* is: *.bitnumber* for a bit in the tag [*element*] for an array element of the tag *.membername* for a structure member of the tag
`MessageType`	Enter Block Transfer Read, Block Transfer Write, CIP Data Table Read, CIP Data Table Write, CIP Generic, PLC2 Unprotected Read, PLC2 Unprotected Write, PLC3 Typed Read, PLC3 Typed Write, PLC3 Word Range Read, PLC3 Word Range Write, PLC5 Typed Read, PLC5 Typed Write, PLC5 Word Range Read, PLC5 Word Range Write, SERCOS IDN Read, SERCOS IDN Write, SLC Typed Read, SLC Typed Write, Unconfigured, or Module Reconfigure. Specify `MessageType := text`
`RemoteElement`	Specify the address or tag name of the element in the remote device. This is the source element of a read instruction or the destination element of a write instruction. Specify `RemoteElement := text`
`RequestedLength`	Specify the number of elements to be transferred (0...32,767). Specify `RequestedLength := value`
`ConnectedFlag`	Specify whether the CIP generic message requires a connection or not. Enter 1 for connected; enter 0 for not connected. Specify `ConnectedFlag := value`
`ConnectionPath`	Specify the connection path to the other device. Specify `ConnectionPath := string`
`CommTypeCode`	Specify the type of communication method. **Enter** **For this communication method** 0 CIP (most messages use CIP communications) 1 DH+ 2 CIP with source ID 3 block transfer via universal remote I/O 4 block transfer via ControlNet Specify `CommTypeCode := value`
`ServiceCode`	If the message type is CIP Generic, specify the service code (0...32,767 hexadecimal). Specify `ServiceCode := 16#value`
`ObjectType`	If the message type is CIP Generic, specify the object type (0...32,767 hexadecimal). The ObjectType attribute is the same as the Class field on the MSG configuration dialog. Specify `ObjectType := 16#value`
`TargetObject`	If the message type is CIP Generic, specify the target object (0...32,767 decimal). The TargetObject attribute is the same as the Instance field on the MSG configuration dialog. Specify `TargetObject := value`
`AttributeNumber`	If the message type is CIP Generic, specify the attribute number (0...65,535 hexadecimal). Specify `AttributeNumber := 16#value`

Attribute	Description
`Channel`	For a DH+ or block transfer message, specify the channel. Enter either A or B. Specify `Channel := value`
`SourceLink`	If the communication method uses DH+, specify the source link (0...199). Specify `DHPlusSourceLink := value`
`DestinationLink`	If the communication method uses DH+, specify the destination link (0...199). Specify `DHPlusDestinationLink := value`
`DestinationNode`	If the communication method uses DH+, specify the destination node number (0...77 octal). Specify `DHPlusDestinationNode := value`
`Rack`	For a DH+ or block transfer message, enter the rack number (0...77 octal) of the target device. Specify `Rack := value`
`Group`	For a DH+ or block transfer message, enter the group number (0...7) of the target device. Specify `Group := value`
`Slot`	For a DH+ or block transfer message, enter the slot number (0...15) of the target device. Specify `Slot := value`
`LocalIndex`	Specify the index into the local element, typically 0. Specify `LocalIndex := value`
`RemoteIndex`	Specify the index into the remote element, typically 0. Specify `RemoteIndex := value`
`LocalElement`	Specify the tag name of the element in the local controller. This is the destination element of a read instruction or the source element of a write instruction. Specify `LocalElement := text`
`DestinationTag`	Specify the tag name of the destination element. Specify `DestinationTag := text`
`CacheConnections`	If the message is to cache connections, enter TRUE. If the message is not to cache connections, enter FALSE. Specify `CacheConnections := text`

Specify Attributes for a MOTION_GROUP Tag

A MOTION_GROUP tag has these attributes.

Attribute	Description
`Description`	Provide information about the tag. Specify `Description := "text"`
`Comment`	Provide information about a tag component. Specify `Comment<specifier> := "text"` Where the *specifier* is: *.bitnumber* for a bit in the tag [*element*] for an array element of the tag *.membername* for a structure member of the tag
`GroupType`	Specify the type of motion group. Enter Warning Enabled or Warning Disabled. Specify `GroupType := text`
`CourseUpdatePeriod`	Specify the coarse update period in milliseconds (500...3200ms). Specify `CourseUpdatePeriod := value`

Attribute	Description
PhaseShift	Specify the phase shift (0...65,535). Specify `PhaseShift := value`
GeneralFaultType	Specify whether an error generates a major fault or a non-major fault. Enter Major Fault or Non Major Fault. Specify `GeneralFaultType := text`
AutoTagUpdate	Enter Disabled or Enabled. Specify `AutoTagUpdate := text`

Specify Attributes for a SAFETY Tag

A safety produced or safety consumed tag (Class = Safety) has these attributes, in addition to the attributes for a standard tag.

Attribute	Description
IncludeConnectionStatus	Specify the connection status of the tag. Must be set to Yes for safety produced or consumed tags. Enter Yes or No. This attribute applies to both safety produced and safety consumed tags. Specify `IncludeConnectionStatus := text`
TimeoutMultiplier	Specify the timeout multiplier (default = 2) for a safety controller system. This value determines the RPIs of time to wait for a packet before declaring a time out. This translates into the number of messages that may be lost before declaring a connection error. A Timeout Multiplier of 1 indicates that no messages may be lost; that is, there must be a packet every RPI interval. A Timeout Multiplier of 2 indicates that 1 message may be lost; that is, as long as a packet is seen in 2 times the RPI, no time-out will occur. Enter a number from 1...4, inclusive. This attribute applies only to safety consumed tags. Specify `TimeoutMultiplier := number`
NetworkDelay Multiplier	Specify the network delay multiplier (default = 100%) for a safety controller system. This value lets you reduce or increase the connection reaction time limit in cases where the transport time of the message is significantly less or more than the RPI. This may be the case when the RPI of an output connection is the same as that of a lengthy task period. Enter a percentage from 10...300, inclusive. This attribute applies only to safety consumed tags. Specify `NetworkDelayMultiplier := number`
ReactionTimeLimit	Specify the connection reaction time limit for a safety controller system. RSLogix 5000 software calculates the connection reaction time limit as a function of the RPI, timeout multiplier, and network delay multiplier. The connection reaction time limit is automatically recalculated if any of the above values change. This attribute applies only to safety consumed tags. Specify `ReactionTimeLimit := number`

Define TAG Initial Values

The `initial_value` format follows the C-language initialization syntax, except that you use square brackets instead of curly brackets. The following table shows some examples of entering initial values.

If the tag is	Enter
single, atomic value	[*Value*]
structure with three members	[*Value1*, *Value2*, *Value3*]
structure with a nested structure	[*Value1*, [*Value2*, *Value3*], *Value4*]
structure with a nested array	[*Value1*, [*ArrayValue1*, *ArrayValue2*], *Value3*]

The initial value for a string value identifies the number of characters in the string and the text string. The string TAG uses this format.

`<tag_name> : STRING := [<number>, 'string_text$00 ... $00'];`

Where:

Item	Identifies
`tag_name`	Name of the string tag.
`STRING`	The STRING data type.
`number`	Number of characters in the string.
`string_text`	Text of the string.
`$00`	The string is padded with $00 to fill its maximum of 82 characters. Each $00 equals one character not used in the string. The entire text string, including the $00 characters, is enclosed in single quotation marks.

For example:

```
TAG
      sourcea_string : STRING  :=
[5,'hello$00$00$00
```

Define a Comment for a TAG Component

The comment attribute of a tag declaration lets you provide information about a component of the tag, such as a specific bit, array element, or structure member.

To add a comment to this operand	Enter
Bit 3 of a tag	COMMENT.3 := "*description*"
Element 8 of an array tag	COMMENT[8] := "*description*"
Preset value of a tag	COMMENT.PRE := "*description*"

TAG Guidelines

Keep these guidelines in mind when defining a tag.

- Tags must be defined after devices (if there are no devices, then after the data types) within the controller body.
- Base tags and aliases can be defined out of order within a tag block.
- You cannot define a second dimension without a first dimension or a third dimension without a second dimension.
- The initial values must comply with the tag type and dimensions.
- Whitespace cannot occur within the initial values or within the type/dimension specifier.

TAG Examples

```
TAG
       bits : MySint := [0];
       dest : INT (RADIX := Decimal) := 0;
       overflow OF bits.MyBit0 (RADIX := Binary);
       source : REAL (RADIX := Exponential) := 0.0;
       timer : TIMER[3] := [[0,0,100],[0,10,100],[0,0,50]];
END_TAG
```

This example shows forced tag data.

```
TAG
       dint_a : DINT (RADIX := Decimal) := 0;
       int_a : INT (RADIX := Decimal) := 0;
       tag_a : UDT_A (ProduceCount := 2) := [0,0],
       TagForceData := [0,0,0,0,1,0,-1,-1,1,0,-72,34];
END_TAG
```

Safety TAG Examples

This example shows a consumed tag in a safety project.

```
safetyConsumed : mypcType (Class := Safety,
       Producer := PeerSafetyController,
       RemoteTag := productCount,
       RemoteFile := 0,
       RPI := 10,
       IncludeConnectionStatus := Yes,
       TimeoutMultiplier := 2,
       NetworkDelayMultiplier := 100,
       ReactionTimeLimit := 29.952) := [[2],[0,0,0]],
       TagForceData :=
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];
```

This example shows a produced tag in a safety project:

```
safetyProduced : mypcType (Class := Safety,
       ProduceCount := 3,
       ProgrammaticallySendEventTrigger := Yes,
       IncludeConnectionStatus := Yes) := [[0],[0,0,0]],
       TagForceData :=
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];
```

ALARM_ANALOG and DIGITAL_ALARM Tag Examples

```
my_alarm : ALARM_DIGITAL (ALMMSG.AM:en-us := "my message",
       Severity := 500,
       MinDurationPRE := 0,
       ProgTime := DT#1970-01-01-00:00:00.000000Z,
       EnableIn := false,
       In := false,
       InFault := false,
       Condition := true,
       AckRequired := true,
       Latched := false,
       ProgAck := false,
       OperAck := false,
       ProgReset := false,
       OperReset := false,
       ProgSuppress := false
       OperSuppress := false,
       ProgUnsuppress := false,
       OperUnsuppress := false,
       ProgDisable := false,
       OperDisable := false,
       ProgEnable := false,
       OperEnable := false,
       AlarmCountReset := false,
       AssocTag1 := "BatchNumber",
       AssocTag2 := "LotNumber",
       AssocTag3 := "PlantNumber",
       AssocTag4 := "ShiftNumber",
       AlarmClass := "tank1",
       HMICmd := "ft command");
```

```
my_alarm2 : ALARM_ANALOG (ALMSG.HH:en-us := "High high alarm message",
       ALMMSG.POS:en-us := "pos alarm message",
       ALMMSG.NEG:en-us := "neg alarm message",
       EnableIn := false,
       InFault := false,
       HHEnabled := true,
       HEnabled := false,
       LEnabled := false,
       LLEnabled := false,
       AckRequired := true,
       ProgAckAll := false,
       OperAckAll := false,
       HHProgAck := false,
       HHOperAck := false,
       HProgAck := false,
       HOperAck := false,
       LProgAck := false,
       LOperAck := false,
       LLProgAck := false,
       LLOperAck := false,
       ROCPosProgAck := false,
       ROCPosOperAck := false,
       ROCNegProgAck := false,
       ROCNegOperAck := false,
       ProgSuppress := false,
       OperSuppress := false,
       ProgUnsuppress := false,
       OperUnsuppress := false,
       ProgDisable := false,
       OperDisable := false,
       ProgEnable := false,
       OperEnable := false,
       AlarmCountReset := false,
       In := 0.0,
       HHLimit := 0.0,
       HHSeverity := 500,
       HLimit := 0.0,
       HSeverity := 500,
       LLimit := 0.0,
       LSeverity := 500,
```

```
LLLimit := 0.0,
LLSeverity := 500,
MinDurationPRE := 0,
Deadband := 0.0,
ROCPosLimit := 0.0,
ROCPosSeverity := 500,
ROCNegLimit := 0.0,
ROCNegSeverity := 500,
ROCPeriod := 0.0,
AssocTag1 := "PlantNumber",
AssocTag2 := "ShiftNumber",
AssocTag3 := "BatchNumber",
AssocTag4 := "LotNumber",
AlarmClass := "tank2",
HMICmd := "ft command");
```

Define a PROGRAM Component

Introduction

This chapter explains the overall structure of the PROGRAM component.

Topic	Page
Define a PROGRAM	95
Specify PROGRAM Attributes	96
Specify PROGRAM Attributes for EquipmentPhase Programs	97
PROGRAM Guidelines	97
PROGRAM Example	98
EquipmentPhase PROGRAM Example	98
Safety PROGRAM Example	99
Export a Source Protected Routine	99

Define a PROGRAM

A PROGRAM component follows this structure.

```
PROGRAM <program_name> [(Attributes)]
       [TAG declaration]
       [ROUTINE declaration]
       [FBD_ROUTINE declaration]
       [ST_ROUTINE declaration]
       [SFC_ROUTINE declaration]
END_PROGRAM
```

Where:

Item	Identifies
program_name	The program.
Attributes	Attributes of the program (such as MAIN or FAULT). Can also contain a description of the program. Enclose in parenthesis. Separate each attribute with a comma (,).
TAG	Program-scoped tags. Follows same format as controller-scoped tags.
ROUTINE	Ladder logic routine for this program.

Item	Identifies
FBD_ROUTINE	Function block diagram routine for this program.
ST_ROUTINE	Structured text routine for this program.
SFC_ROUTINE	Sequential function chart routine for this program.

You can intermix ROUTINE, FBD_ROUTINE, ST_ROUTINE, and SFC_ROUTINE declarations.

The maximum number of programs depends on the type of controller.

Controller	Maximum Number of Programs
ControlLogix	100 (32 in firmware revisions prior to 15)
SoftLogix5800	100 (32 in firmware revisions prior to 15)
FlexLogix	32
CompactLogix (1768 and 1769)	32
DriveLogix	32

Specify PROGRAM Attributes

Specify these attributes for a PROGRAM.

Attribute	Description
Description	Provide information about the program. Specify `Description := "text"`
Type	Specify the type of program. If this program is a Equipment Phase program, enter EquipmentPhase. Otherwise, enter Normal. Specify `Type := text`
Class	Specify the class of the program. This attribute applies only to safety controller projects. Enter Standard or Safety. Do not use this attribute if the program is an Equipment Phase program (Type = EquipmentPhase). Specify `Class := text`
Main	Name of the main routine of the program (40 characters maximum). Specify `Main := name`
Fault	Name of the program fault routine, if any (40 characters maximum). Specify `Fault := name`
Mode	Enter 0 for not testing edits; enter 1 for testing edits. Specify `Mode := value`
DisableFlag	Enter 1 to disable the program; enter 0 to enable the program. Specify `DisableFlag := value`
SynchronizeRedundancyDataAfterExecution	Enter 1 to synchronize data after the program scan in a redundant system; enter 0 to not synchronize data after the program scan. Specify `SynchronizeRedundancyDataAfterExecution := value`

Specify PROGRAM Attributes for EquipmentPhase Programs

In addition to the PROGRAM connection attributes previously described, specify these attributes for an Equipment Phase program (PROGRAM Type = EquipmentPhase).

Attribute	Description
`Prestate`	Name of the prestate routine (40 characters maximum). Specify `Prestate := name`
`InitialStepIndex`	Specify an integer value for the initial step index of the phase. Specify `InitialStepIndex := number`
`InitialState`	Specify state of the phase. Enter Aborted, Completed, Stopped, or Idle (default). Specify `InitialState := text`
`CompleteStateIfNot Impl`	If the phase does not implement all the expected states, enter StateComplete (default) so the program can continue to execute when it expects a state that was omitted. The program ignores the omitted state and continues to the next state. Otherwise, enter NoAction. Specify `CompleteStateIfNotImpl := text`
`LossOfCommCmd`	If the phase uses an external sequencer, such as RSBizWare Batch software, specify that appropriate action to take if communication fails between the controller and the external sequencer. Enter Abort, Hold, Stop or none (default). Specify `LossOfCommCmd := text`
`ExternalRequest Action`	Specify how to handle an external request (PXRQ instruction) that is in process when the phase receives the command to go to a Holding state. Enter Clear to abort outstanding external requests. Otherwise, enter none (default). Specify `ExternalRequestAction := text`
`EquipmentId`	The RSBizware Batch equipment identifier for the Equipment Phase. This value is set by the RSBizware Batch software when you synchronize with an RSLogix 5000 project file. Do not modify this value.
`RecipePhaseNames`	The RSBizware Batch recipe phases for the Equipment Phase. This value is set by the RSBizware Batch software when you synchronize with an RSLogix 5000 project file. Do not modify this value.

PROGRAM Guidelines

Keep in mind these guidelines when defining a program.

- The MAIN and FAULT attributes can be defined in any order.
- The TAG declaration block must occur before the routine block.
- All tag collection declaration blocks that occur in a program definition block are imported as tags of a given program and can be only seen by routines under that program. Controller tags, on the other hand can be seen by routines in any program.

PROGRAM Example

```
PROGRAM Prg1 (Main := RoutineB, Description := "I $'am$'" "
$0034 a $"program$"")
       TAG
       st11 : DINT (RADIX := Decimal, ProduceCount := 0) := 2;
       st12 : BOOL (RADIX := Binary, ProduceCount := 0) :=
2#00000000;
       END_TAG

       ROUTINE RoutineA
       JSR(_2_LADDER, 0);
       END_ROUTINE

       ROUTINE RoutineB
              RC: "$L ** ;MORE $";STUFF" do not include "more";
              xic(st11) ote(st12);
       END_ROUTINE
END_PROGRAM
```

EquipmentPhase PROGRAM Example

```
PROGRAM Add_Water (Type := EquipmentPhase,
       PreState := Prestate_Routine,
       FAULT := Fault_Routine,
       MODE := 0,
       DisableFlag := 0,
       InitialStepIndex := 1,
       InitialState := Idle,
       CompleteStateIfNotImpl := StateComplete,
       LossOfCommCmd := None,
       ExternalRequestAction := None)

       [TAG declaraions]

       [ROUTINE declaraions]

       [SFC_ROUTINE declaraions]

END_PROGRAM
```

Safety PROGRAM Example

```
PROGRAM SafetyProgram (Class := Safety,
       MAIN := MainRoutine,
       MODE := 0,
       DisableFlag := 0)
       TAG
              baseballs : TIMER  := [0,0,0];
              basketballs : BOOL (RADIX := Decimal) := 0;
              footballs : DINT (RADIX := Decimal) := 0;
              golfballs : LIGHT_CURTAIN  :=
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];
              hockeypucks : mypcType  := [[0],[0,0,0]];
              in : DINT (RADIX := Decimal) := 0;
              sourceTag : LIGHT_CURTAIN  :=
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];
       END_TAG

       ROUTINE MainRoutine
              N: XIC(Safety1.4)OTE(aliasToSafety.5);
              N:
LC(sourceTag,AUTOMATIC,in.0,in.1,10,in.2,in.3,in.4);
       END_ROUTINE

END_PROGRAM
```

Export a Source Protected Routine

If the project contains a source-protected routine and the key is not available when you export the project, the routine appears within an ENCODED_DATA component.

```
ENCODED_DATA <EncodedType> <name> [(Attributes)]
END_ENCODED_DATA
```

Where:

Item	Identifies
`EncodedType`	The type of source-protected logic.
`name`	The name of the routine.
`Attributes`	Other attributes of the routine that are not controlled via source protection.

Source Protected Example

IMPORTANT When the source is protected, the information between the ENCODED_DATA and END_ENCODED_DATA statements is encrypted. If you modify this encrypted information in any way, you will not be able to re-import the information.

```
ENCODED_DATA (EncodedType := ST_ROUTINE,
                      Name := "Example")
xbr+GxxsZCMWfkzJFlY7U7CmecNwRm1oF+u+QYVIAy6R6eLxaznX+7SvV8IoQO9EBjo7LzK+N0C9OMYDT9vrJAX2DdWcP
RnCnd2bjNopNxDC9XZdRVz9x2vT8rRZSIFz69asXJRvrhVZTCBee0sO6dWpYtvqrHfLV/4hOFJFt3mB9qQTN+oWe589Kd
Dw5Iz21YwT3UtlkeE8KP4b+7iwRYLmAOydRb/ygqTE01zDpZAS9AlcebPoYUhM1gCF1AuK9eYBk0q/HI4lpDsO/VVvi7V
d7U+4/xPPRqiEjoDI25pjFZwZu4UaGvf4jQK0NQj2wPcnu4pCbzfkNMJBPCZdfe61H4v19B7rPwxfd9Q86rl26Bu0gLeD
yllG6Suoxa8OYersEhBLz6nrTYd/wybHvNmp0TLgpI1bhzWyTVttWf4ZW8qBN6Yu796cGwaf/f5kXQ5bUni6eys/vIV/G
FVQ48Z3Vg+SWC3KUawvBIhbIhvrB1NFgcPVX7otxeGByvN5mtsVLSKw7paGvIL6n4jLC8mNJ0LuBdPtZqI5nxSdbNSKqz
hD+AqoCNtIrKoFRc1y85xGYaVV7X1HmhOnDQqokX3xPquwBiNByl5nJ4xNg91QACSC8Xm7tQa0IPhQKfAdvQWKDVAGfH9
xJarokPtx0ezxcPKut/cm4hmBmfWxyeAoZfN01cNMtaNBdYSXe1qVv0h6tiPOhKPAyaJA7Mdg2zSRvCviwBDSDs8OAIyk
YyEHy0J0VqB1nj3eFn5e8eFa8qBvmtQhvP7VizHPN5694c8Br/+tE7ZXIcq7UzD8r+evWQqeu2wuwbCn8L1/M7wEnEkKe
YzX+dA2Un7pK3Xuc3ABUcVOJFV3A+ZiqLGkyCJYwrGVS+OjWPu/+QauyCkkH4dd2IBcWBKgo22v1potIC7wrBjHbbeKMN
tLlAsofwX4bKjcuWp/lXTuwu20B78GV2+W+HJu2ZsBVvVRPLa+A+ueWZhVqAsOsMO5q1p/Z5hDGWJqEwbcDVeBYgCc+n5
usIr5J1OeHh9gPbkQz/KYWPY4XcyS2dqmWzn3rYJ/2GAbJrEJwfYZl1rV+NBeUDbvR0biyGlrcbIlsq/OlLqiXcAnMOGq
5jW/JR8gn3uylWTuYo0wSimnSNEC82cmffZkg5RRpD5C5ySp7go2zkVNBLuzSRTfTUN4/S9xYptSod7fGdp1sQ3IhZ3rK
vz+7ifr3y9Mhwgxi7WXPOZ+0lgi/YJXLuhEv09g5lEpTv/hJ8xXeH8dNyNZ7E0O8HdNDl00XQ

END_ENCODED_DATA
```

Enter Ladder Diagram Logic

Introduction

This chapter explains the how to enter ladder diagram logic in a complete import/export file.

Topic	Page
Enter a Ladder Logic Routine	101
Enter Rung Logic	102
Enter Rung Comments	104
Enter Neutral Text for Ladder Instructions	104

Enter a Ladder Logic Routine

A ladder logic ROUTINE follows this structure.

```
ROUTINE <routine_name> [Attributes]
       <ladder rungs>
END_ROUTINE
```

Where:

Item	Identifies
`routine_name`	The routine.
`Attributes`	Attributes of the routine. Can also contain a description of the routine. Separate each attribute with a comma (,).
`ladder rungs`	Ladder logic.

Specify ROUTINE Attributes

Specify these attributes for a ROUTINE.

Attribute	Description
`Description`	Provide information about the routine. Specify `Description := "text"`

Enter Rung Logic

Enter rung logic within a ROUTINE component in an import/export file. Each rung follows this structure.

```
<RungType> : <RungNeutralText>;
```

Where:

Item	Identifies
`RungType`	The rung.
`RungNeutralText`	The logic.

The following rung types are available.

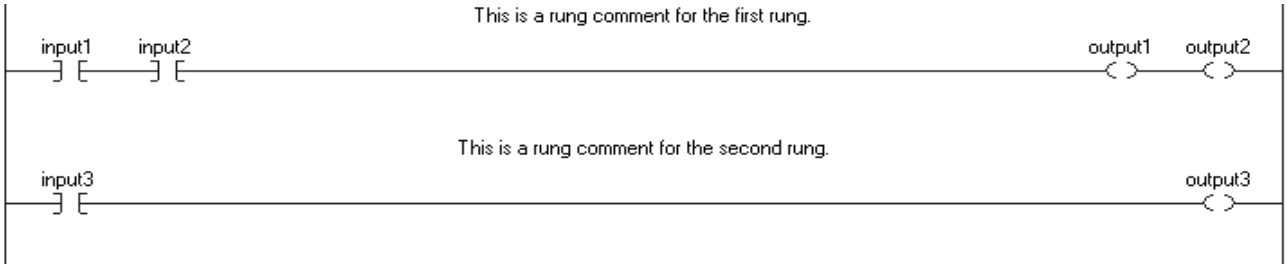
Item	Identifies
N	Normal
I	Insert
D	Delete
IR	Insert with a replace
rR	Pending replace IR
R	Replace
rI	Pending replace I
rN	Pending replace N
e	Pending insert rung
er	Pending replace rung

Rung Guidelines

- Rungs are specified using neutral language. See the rest of this chapter for the neutral text language format for the supported instructions.
- Each rung ends with a semicolon (;).

Ladder ROUTINE Example

This ladder routine exports as shown below.




```
ROUTINE Ladder_example
       RC: "This is a rung comment for the first rung.";
       N: XIC(input1)XIC(input2)OTE(output1)OTE(output2);
       RC: "This is a rung comment for the second rung.";
       N: XIC(input3)OTE(output3);
END_ROUTINE
```

Enter Branches

Enter a single branch or simultaneous branches on a rung. A branch follows this structure.

```
[,BranchNeutralText]
```

Where:

Item	Identifies
[ ]	The branch.
,	The beginning of each branch within the branch, to account for simultaneous branches.
space	The end of each branch within the branch, to account for simultaneous branches.
BranchNeutralText	The logic.

Example with a Single Branch

```
N: XIC(conveyor_a)[,XIC(input_1) XIO(input_2) ]OTE(light_1);
```

Example with Two Simultaneous Branches

```
N: XIC(conveyor_b)[,XIC(input_1) XIO(input_2) ,XIC(input_a) XIO(input_b)  ]OTE(light_2);
```

Enter Rung Comments

The comments for rungs use this syntax.

```
RC: “comment” “more” “etc”;
```

A rung comment must be followed by a rung.

Enter Neutral Text for Ladder Instructions

The following tables lists each ladder instruction and its neutral text format. For details about a specific instruction, see one of these manuals.

Instruction Type	Resource
Basic, sequential instruction	Logix5000 Controllers General Instructions Set Reference Manual, publication 1756-RM003
Process control or drives instruction	Logix5000 Controllers Process Control and Drives Instruction Set Reference Manual, publication 1756-RM006
Motion instruction	Logix5000 Controllers Motion Instructions Set Reference Manual, publication 1756-RM007

Instruction	Neutral Text Format
ABL	`ABL(channel,serial_port_control,character_count);`
ABS	`ABS(source,destination);`
ACB	`ACB(channel,serial_port_control,character_count);`
ACL	`ACL(channel,clear_serial_port_read,clear_serial_port_write);`
ACS	`ACS(source,destination);`
ADD	`ADD(source_A,source_B,destination);`
AFI	`AFI();`
AHL	`AHL(channel,ANDMask,ORMask,serial_port_control,channel_status);`
ALMA	`ALMA (alma_tag,in,program_acknowledge_all,program_disable,program_enable);`
ALMD	`ALMD (almd_tag,program_acknowledge,program_reset,program_disable, program_enable);`
AND	`AND(source_A,source_B,destination);`

Instruction	Neutral Text Format
ARD	ARD(*channel,destination,serial_port_control,string_length, characters_read*);
ARL	ARL(*channel,destination,serial_port_control,string_length, characters_read*);
ASN	ASN(*source,destination*);
ATN	ATN(*source,destination*);
AVE	AVE(*array,dim_to_vary,destination,control,length,position*);
AWA	AWA(*channel,source,serial_port_control,string_length,characters_sent*);
AWT	AWT(*channel,source,serial_port_control,string_length,characters_sent*);
BRK	BRK();
BSL	BSL(*array,control,source_bit,length*);
BSR	BSR(*array,control,source_bit,length*);
BTD	BTD(*source,source_bit,destination,destination_bit,length*);
CLR	CLR(*destination*);
CMP	CMP(*expression*);
CONCAT	CONCAT(*sourceA,sourceB,destination*)
COP	COP(*source,destination,length*);
COS	COS(*source,destination*);
CPS	CPS(*source,destination,length*)
CPT	CPT(*destination,expression*);
CTD	CTD(*counter,preset,accum*);
CTU	CTU(*counter,preset,accum*);
DDT	DDT(*source,reference,result,cmp_control,length,position,result_control, length,position*);
DEG	DEG(*source,destination*);
DELETE	DELETE(*source,quantity,start,destination*);
DIN	DIN(*din_tag,reset_type,channel_A,channel_B,circuit_reset,fault_reset*);
DIV	DIV(*source_A,source_B,destination*);
DTOS	DTOS(*source,destination*);
DTR	DTR(*source,mask,reference*);
ENPEN	ENPEN(*enpen_tag,reset_type,channel_A,channel_B,circuit_reset,fault_reset*);
EOT	EOT(*data_bit*);
EQU	EQU(*source_A,source_B*);
ESTOP	ESTOP(*estop_tag,reset_type,channel_A,channel_B,circuit_reset,fault_reset*);
EVENT	EVENT(*task*);
FAL	FAL(*control,length,position,mode,destination,expression*);
FBC	FBC(*source,reference,result,cmp_control,length,position,result_control, length,position*);

Instruction	Neutral Text Format
FFL	FFL(*source,FIFO,control,length,position*);
FFU	FFU(*FIFO,destination,control,length,position*);
FIND	FIND(*source,search,start,result*);
FLL	FLL(*source,destination,length*);
FOR	FOR(*routine_name,index,initial_value,terminal_value,step_size*);
FPMS	FPMS(*fpms_tag,input_1,input_2,input_3,input_4,input_5,fault_reset*);
FRD	FRD(*source,destination*);
FSC	FSC(*control,length,position,mode,expression*);
GEQ	GEQ(*source_A,source_B*);
GRT	GRT(*source_A,source_B*);
GSV	GSV(*class_name,instance_name,attribute_name,destination*);
INSERT	INSERT(*sourceA,sourceB,start,destination*);
IOT	IOT(*output_tag*);
JMP	JMP(*label_name*);
JSR	JSR(*routine_name,input_1,...input_n,return_1,..return_n*);
JXR	JXR(*external_routine_name,external_routine_control,parameter, return_parameter*);
LBL	LBL(*label_name*);
LC	LC(*lc_tag,reset_type,channel_A,channel_B,input_filter_time, mute_light_curtain,circuit_reset,fault_reset*);
LEQ	LEQ(*source_A,source_B*);
LES	LES(*source_A,source_B*);
LFL	LFL(*source,LIFO,control,length,position*);
LFU	LFU(*LIFO,destination,control,length,position*);
LIM	LIM(*low_limit,test,high_limit*);
LN	LN(*source,destination*);
LOG	LOG(*source,destination*);
LOWER	LOWER(*source,destination);*
MAAT	MAAT(*axis,motion_control*);
MAFR	MAFR(*axis,motion_control*);
MAG	MAG(slave*_axis,master_axis,motion_control,direction,ratio,slave_counts,master_counts,master_reference,ratio_format,clutch,accel_rate,accel_units*);
MAH	MAH(*axis,motion_control*);
MAHD	MAHD(*axis,motion_control,diagnostic_test,observed_direction*);
MAJ	MAJ(*axis,motion_control,direction,speed,speed_units,accel_rate, accel_units,decel_rate,decel_units,profile,merge,merge_speed*);
MAM	MAM(*axis,motion_control,move_type,position,speed,speed_units,accel_rate, accel_units,decel_rate,decel_units,profile,merge,merge_speed*);

Instruction	Neutral Text Format
MAOC	MAOC(*axis,execution_target,motion_control,output,input,output_cam, cam_start_position,cam_end_position,output_compensation,execution_mode, execution_schedule,axis_arm_position,cam_arm_position,reference*);
MAPC	MAPC(*slave_axis,master_axis,motion_control,direction,cam_profile, slave_scaling,master_scaling,execution_mode,execution_schedule, master_lock_position,cam_lock_position,master_reference, master_direction*);
MAR	MAR(*axis,motion_control,trigger_condition,windowed_registration, minimum_position,maximum_position*);
MAS	MAS(*axis,motion_control,stop_type,change_decel,decel_rate,decel_units*);
MASD	MASD(*axis,motion_control*);
MASR	MASR(*axis,motion_control*);
MATC	MATC(*axis,motion_control,direction,cam_profile,distance_scaling, time_scaling,execution_mode,execution_schedule*);
MAW	MAW(*axis,motion_control,trigger_condition,position*);
MCCD	MCCD(*coordinate_system,motion_control,motion_type,change_speed,speed, speed_units,change_accel,accel_rate,accel_units,change_decel,decel_rate, decel_units,scope*);
MCCM	MCCM(*coordinate_system,motion_control,move_type,position,circle_type, via/center/radius,direction,speed,speed_units,accel_rate,accel_units, decel_rate,decel_units,profile,termination_type,merge,merge_speed*);
MCCP	MCCP(*motion_control,cam,length,start_slope,end_slope,cam_profile*);
MCLM	MCLM(*coordinate_system,motion_control,move_type,position,speed, speed_units,accel_rate,accel_units,decel_rate,decel_units,profile, termination_type,merge,merge_speed*);
MCD	MCD(a*xis,motion_control,motion_type,change_speed,speed,change_accel, accel_rate,change_decel,decel_rate,speed_units,accel_units, decel_units*);
MCR	MCR();
MCS	MCS(*coordinate_system,motion_control,stop_type,change_decel,decel_rate, decel_units*);
MCSD	MCSD(*coordinate_system,motion_control*);
MCSR	MCSR(*coordinate_system,motion_control*);
MCSV	MCSV(*motion_control,cam_profile,master_value,slave_value,slope_value, slope_derivative*);
MCT	MCT(*source_system,target_system,motion_control,orientation,translation*);
MCTP	MCTP(*source_system,target_system,motion_control,orientation,translation, transform_direction,reference_position,transform_position*);
MDF	MDF(*axis,motion_control*);
MDO	MDO(*axis,motion_control,drive_output,drive_units*);
MDOC	MDOC(*axis,execution_target,motion_control,disarm_type*);
MDR	MDR(*axis,motion_control*);
MDW	MDW(*axis,motion_control*);
MEQ	MEQ(*source,mask,compare*);

Instruction	Neutral Text Format
MGS	`MGS(group,motion_control,stop_mode);`
MGSD	`MGSD(group,motion_control);`
MGSP	`MGSP(group,motion_control);`
MGSR	`MGSR(group,motion_control);`
MID	`MID(source,quantity,start,destination);`
MOD	`MOD(source_A,source_B,destination);`
MOV	`MOV(source,destination);`
MRAT	`MRAT(axis,motion_control);`
MRHD	`MRHD(axis,motion_control,diagnostic_test);`
MRP	`MRP(axis,motion_control,type,position_select,position);`
MSF	`MSF(axis,motion_control);`
MSG	`MSG(message_control);`
MSO	`MSO(axis,motion_control);`
MUL	`MUL(source_A,source_B,destination);`
MVM	`MVM(source,mask,destination);`
NEG	`NEG(source,destination);`
NEQ	`NEQ(source_A,source_B);`
NOP	`NOP();`
NOT	`NOT(source,destination);`
ONS	`ONS(storage_bit);`
OR	`OR(source_A,source_B,destination);`
OSF	`OSF(storage_bit,output_bit);`
OSR	`OSR(storage_bit,output_bit);`
OTE	`OTE(data_bit);`
OTL	`OTL(data_bit);`
OTU	`OTU(data_bit);`
PATT	`PATT(phase_name,result);`
PCLF	`PCLF(phase_name);`
PCMD	`PCMD(phase_name,command,result);`
PDET	`PDET(phase_name);`
PFL	`PFL(source);`
PID	`PID(PID,process_variable,tieback,control_variable,pid_master_loop, inhold_bit,inhold_value);`
POVR	`POVR(phase_name,command,result);`
PPD	`PPD();`
PRNP	`PRNP();`
PSC	`PSC();`

Instruction	Neutral Text Format
PXRQ	PXRQ(*phase_instruction,external_request,data_value*);
RAD	RAD(*source,destination*);
RES	RES(*structure*);
RET	RET(*return_1,...return_n*);
RIN	RIN(*rin_tag,reset_type,channel_A,channel_B,circuit_reset,fault_reset*);
ROUT	ROUT(*rout_tag,feedback_type,enable,feedback_1,feedback_2,fault_reset*);
RTO	RTO(*timer,preset,accum*);
RTOS	RTOS(*source,destination*)
SBR	SBR(*routine_name,input_1,...input_n*);
SFP	SFP(SFC_routine_name,target_state);
SFR	SFR(SFC_routine_name,step_name);
SIN	SIN(*source,destination*);
SIZE	SIZE(*souce,dimension_to_vary,size*);
SQI	SQI(*array,mask,source,control,length,position*);
SQL	SQL(*array,source,control,length,position*);
SQO	SQO(*array,mask,destination,control,length,position*);
SQR	SQR(*source,destination*);
SRT	SRT(*array,dim_to_vary,control,length,position*);
SSV	SSV(*class_name,instance_name,attribute_name,source*);
STD	STD(*array,dim_to_vary,destination,control,length,position*);
STOD	STOD(*source,destination*)
STOR	STOR(*source,destination*)
SUB	SUB(*source_A,source_B,destination*);
SWPB	SWPB(*source,order_mode,destination*);
TAN	TAN(*source,destination*);
THRS	THRS(*thrs_tag,active_pin_type,active_pin,right_button_normally_open, right_button_normally_closed,left_button_normally_open, left_button_normally_closed,fault_reset*);
TND	TND();
TOD	TOD(*source,destination*);
TOF	TOF(*timer,preset,accum*);
TON	TON(*timer,preset,accum*);
TRN	TRN(*source,destination*);
UID	UID();
UIE	UIE();
UPPER	UPPER(*source,destination);*
XIC	XIC(*data_bit*);

Instruction	Neutral Text Format
XIO	`XIO(data_bit);`
XOR	`XOR(source_A,source_B,destination);`
XPY	`XPY(source_A,source_B,destination);`

Enter Function Block Diagram Logic

Introduction

This chapter explains the how to enter function block diagram logic in a complete import/export file.

Topic	Page
Enter a Function Block Diagram Routine	111
Enter Function Block Diagram Logic	112
Enter IREFs and OREFs	117
Enter ICONs and OCONs	119
Enter Wires and Feedback Wires	120
Enter Blocks	121
Enter Text Boxes	123
Enter Attachments	124
Enter Parameters for Function Block Instructions	125

Enter a Function Block Diagram Routine

A function block FBD_ROUTINE follows this structure.

```
FBD_ROUTINE <routine_name> [Attributes]
       <function block sheets>
END_FBD_ROUTINE
```

Where:

Item	Identifies
`routine_name`	The routine.
`Attributes`	Attributes of the function block routine (such as sheet Size or sheet orientation). Can also contain a description of the routine. Separate each attribute with a comma (,).
`function block sheets`	Enter function block logic in sheets.

Specify FBD_ROUTINE Attributes

Specify these attributes for a FBD_ROUTINE.

Attribute	Description
`Description`	Provide information about the routine. Specify `Description := "text"`
`SheetSize`	Select one of these sizes. • Letter (8.5x11in) • Legal (8.5x14in) • Tabloid (11x17.in) • A4 (210x297mm) • A3 (297x420mm) Specify `SheetSize := size`
`SheetOrientation`	Select the orientation of the sheet as Portrait or Landscape. Specify `SheetOrientation := type`

Enter Function Block Diagram Logic

Enter function block diagram logic in sheets within a FBD_ROUTINE component in an import/export file.

```
(* sheet <sheet_number> *)
SHEET (Name := <sheet_name>)
       <IREF_component>
       <ICON_component>
       <mnemonic_BLOCK_componment>
       <OREF_component>
       <OCON_component>
       <WIRE_component>
       <FEEDBACK_WIRE_component>
END_SHEET
```

Where:

Item	Identifies
`Name`	The name of the sheet. Specify `Name := "text"`
`IREF`	Input references.
`ICON`	Input wire connectors.
`mnemonic_BLOCK`	Function block instructions and their locations.
`OREF`	Output references.
`OCON`	Output wire connectors.
`WIRE`	Wires and what they are attached to.
`FEEDBACK_WIRE`	Feedback wires and what they are attached to.

SHEET Guidelines

- The sheets in the routine appear in order in the export file. Each sheet section contains all the drawing elements and wires for that sheet.
- The sheet number is stored in a comment at the beginning of the sheet for reference only. On import, sheet numbers are assigned based on order in the file, not on the number in the comment.
- The sheet name is stored as an attribute because it is optional.
- Input references, blocks, output references, special drawing elements, and wires are contained within the sheet. On export, the elements appear in the order shown. On import, elements can be interspersed in the file.
- WIRE and FEEDBACK_WIRE statements must appear after all the other components.
- Be careful when copying and pasting function block components within an import/export file. Each component within a sheet must have a unique ID number within that sheet.

FBD_ROUTINE Example

```
FBD_ROUTINE My_FBD_Routine (SheetSize := "Tabloid (11x17in)", SheetOrientation := Landscape)
       SHEET  (Name := Input_Scaling)
              MUL_BLOCK  (ID := 0,
                          X := 440,
                          Y := 60,
                          Operand := MUL_01,
                          VisiblePins := "SourceA, SourceB, Dest")
              END_MUL_BLOCK

              SCL_BLOCK  (ID := 1,
                          X := 240,
                          Y := 60,
                          Operand := SCL_01,
                          VisiblePins := "In, InEUMax, Out, MaxAlarm")
              END_SCL_BLOCK

              PI_BLOCK  (ID := 2,
                         X := 260,
                         Y := 260,
                         Operand := PI_01,
                         VisiblePins := "In, Initialize, InitialValue, Out, HighAlarm,
LowAlarm")
              END_PI_BLOCK

              IREF  (ID := 3,
                     X := 120,
                     Y := 120,
                     Operand := Input_Tag)
              END_IREF

              ICON  (ID := 4,
                     X := 160,
                     Y := 320,
                     Name := ConnectorName)
              END_ICON
```

```
OREF  (ID := 5,
       X := 520,
       Y := 320,
       Operand := Output_Tag)
END_OREF

OCON  (ID := 6,
       X := 680,
       Y := 100,
       Name := ConnectorName)
END_OCON

FEEDBACK_WIRE  (FromElementID := 0,
                FromParameter := Dest,
                ToElementID := 0,
                ToParameter := SourceB)
END_FEEDBACK_WIRE

WIRE  (FromElementID := 3,
       FromParameter := "",
       ToElementID := 1,
       ToParameter := In)
END_WIRE

WIRE  (FromElementID := 4,
       FromParameter := "",
       ToElementID := 2,
       ToParameter := In)
END_WIRE

WIRE  (FromElementID := 0,
       FromParameter := Dest,
       ToElementID := 6,
       ToParameter := "")
END_WIRE
```

```
            WIRE  (FromElementID := 1,
                   FromParameter := Out,
                   ToElementID := 0,
                   ToParameter := SourceA)
            END_WIRE

            WIRE  (FromElementID := 2,
                   FromParameter := Out,
                   ToElementID := 5,
                   ToParameter := "")
            END_WIRE

      END_SHEET

END_FBD_ROUTINE
```

Export Function Block Logic While Editing Online

If you export function block logic that contains online edits, the export file exports LOGIC blocks to indicate the original, test edits, and pending edits states. If there are no online edits, you will not see these LOGIC blocks.

Example 1: Both Test Edits and Pending Edits Exist

```
FBD_ROUTINE MyFbdRoutine (SheetSize := "Letter (8.5x11in)", SheetOrientation := Landscape)
      LOGIC (Online_Edit_Type := Orig)
            (* Sheets inserted here - see format described above *)
      END_LOGIC

      LOGIC (Online_Edit_Type := Test)
            (* Sheets inserted here - see format described above *)
      END_LOGIC

      LOGIC (Online_Edit_Type := Pend)
            (* Sheets inserted here - see format described above *)
      END_LOGIC
END_FBD_ROUTINE
```

Example 2: Only Pending Edits Exist

```
FBD_ROUTINE MyFbdRoutine (SheetSize := "Letter (8.5x11in)", SheetOrientation := Landscape)
       LOGIC (Online_Edit_Type := Orig)
              (* Sheets inserted here - see format described above *)
       END_LOGIC

       LOGIC (Online_Edit_Type := Pend)
              (* Sheets inserted here - see format described above *)
       END_LOGIC
END_FBD_ROUTINE
```

Where:

Item	Identifies
`Online_Edit_Type`	Whether online edits exist when the logic is exported. If online edits exist, there will be a LOGIC block for Online_Edit_Type := Orig and then the appropriate LOGIC block for the existing edits. Online_Edit_Type : = Pend indicates pending edits. Online_Edit_Type := Test indicates test edits. If there are no online edits when the logic is exported, there are no LOGIC blocks and the main components in the routine are SHEET components.

Enter IREFs and OREFs

Input and output references have similar formats and identical attributes.

```
IREF (
       ID := <unique_identifier>
       X := <internal_grid_x_location>,
       Y := <internal_grid_y_location>,
       Operand := <tag_reference>)
END_IREF

OREF (
       ID := <unique_identifier>
       X := <internal_grid_x_location>,
       Y := <internal_grid_y_location>,
       Operand := <tag_reference>)
END_OREF
```

Where:

Item	Identifies
`ID`	The IREF or OREF identifier; uniqueness is important for wiring. Enter an unsigned, 32-bit integer value. Specify `ID := "number"`
`X`	X-coordinates on internal grid. Enter an unsigned, 32-bit integer value. Specify `X := "number"`
`Y`	Y-coordinates on internal grid. Enter an unsigned, 32-bit integer value. Specify `Y := "number"`
`Operand`	The reference (optional). Enter a tag or literal value for IREF; enter tag for OREF. Specify `Operand := "tag"`

IREF and OREF Guidelines

- If the Operand is not a qualified tag or literal value, the IREF/OREF will not be verified.
- The X and Y grid locations are a relative position from the upper-left corner of the sheet. X is the horizontal position; Y is the vertical position.

IREF and OREF Examples

```
IREF  (ID := 8,
       X := 200,
       Y := 380,
       Operand := PMUL_InitVal)
END_IREF


OREF  (ID := 9,
       X := 480,
       Y := 340,
       Operand := FB_PMUL)
END_OREF
```

Enter ICONs and OCONs

Input and output wire connectors have similar formats and identical attributes.

```
ICON (ID := <unique_identifier>
       X := <internal_grid_x_location>,
       Y := <internal_grid_y_location>,
       Name := <connector_name>)
END_ICON

OCON (ID := <unique_identifier>
       X := <internal_grid_x_location>,
       Y := <internal_grid_y_location>,
       Name := <connector_name>)
END_OCON
```

Where:

Item	Identifies
`ID`	The ICON or OCON identifier; uniqueness is important for wiring. Enter an unsigned, 32-bit integer value. Specify `ID := "number"`
`X`	X-coordinates on internal grid. Enter an unsigned, 32-bit integer value. Specify `X := "number"`
`Y`	Y-coordinates on internal grid. Enter an unsigned, 32-bit integer value. Specify `Y := "number"`
`Name`	The name of the wire connector (optional). Specify `Name := "number"`

ICON and OCON Guidelines

- OCON connector names must be unique within a function block routine.
- Multiple ICON connector names can reference the same OCON connector name.
- ICONs and OCONs with unmatched or blank connector names will not be verified.
- The X and Y grid locations are a relative position from the upper-left corner of the sheet. X is the horizontal position; Y is the vertical position.

ICON and OCON Examples

```
ICON  (ID := 1,
       X := 140,
       Y := 300,
       Name := MyConnector)
END_ICON

OCON  (ID := 4,
       X := 460,
       Y := 140,
       Name := MyConnector)
END_OCON
```

Enter Wires and Feedback Wires

The wire and feedback wire formats describe a wire by specifying what it is attached to at each end, which is always a pin on another drawing element. Wires and feedback wires follow this format:

```
WIRE
       FromElementID := <identifier_of_from_element>,
       FromParameter := <name_of_output_pin>,
       ToElementID := <identifier_of_to_element>,
       ToParameter := <name_of_input_pin>
END_WIRE
```

Where:

Item	Identifies
`FromElementID`	The source drawing element. Enter an unsigned, 32-bit integer. Specify `FromElementID := "number"`
`FromParameter`	The pin on the source drawing element. **For** — **Enter** Blocks — Parameter name IREFs — In ICONs — In Specify `FromParameter := "pin"`
`ToElementID`	The destination drawing element. Enter an unsigned, 32-bit integer. Specify `ToElementID := "number"`
`ToParameter`	The pin on the destination drawing element. **For** — **Enter** Blocks — Parameter name OREFs — Out OCONs — Out Specify `ToParameter := "pin"`

WIRE Guidelines

- Wires that are not correctly specified will not be imported.
- A feedback wire follows the same format as a wire. Just connect the source and destination elements to form a feedback.

WIRE Example

```
WIRE  (FromElementID := 11,
       FromParameter := "",
       ToElementID := 1,
       ToParameter := Initialize)
END_WIRE
```

Enter Blocks

All function blocks follow this format.

```
mnemonic_BLOCK (
       ID := <unique_identifier>
       X := <internal_grid_x_location>,
       Y := <internal_grid_y_location>,
       Operand := <block_tag_reference>,
       <Array_Name>Operand := <array_tag_reference>,
       VisiblePins := "<parameter_name>, …")
END_mnemonic_BLOCK
```

Where:

Item	Identifies
`ID`	The block identifier; uniqueness is important for wiring. Enter an unsigned, 32-bit integer value. Specify `ID := "number"`
`X`	X-coordinates on internal grid. Enter an unsigned, 32-bit integer value. Specify `X := "number"`
`Y`	Y-coordinates on internal grid. Enter an unsigned, 32-bit integer value. Specify `Y := "number"`

Item	Identifies
`Operand`	Tag name for the block (optional). Specify `Operand := "tag_name"`
`ArrayName`	Tag name for array (optional). Specify `ArrayName := "array_name"`
`VisiblePins`	Comma-separated list of the names of all the parameters with pins visible for wiring. The names match the member names of the data type of the block tag. Specify `VisiblePins := "parameter"`

BLOCK Guidelines

- If the Operand is not a qualified tag of the correct data type, the block will not be verified.
- Some function block instructions require specific arrays. This table lists the valid Array Name for each of these instructions.

Instruction	Array Name
DEDT	Storage (required)
FGEN	X1 (required) Y1 (required) X2 (optional) Y2 (optional)
MAVE	Storage (required) Weight (optional)
RMPS	RampValue (required) SoakValue (required) SoakTime (required)

- The X and Y grid locations are relatives position from the upper-left corner of the sheet. X is the horizontal position; Y is the vertical position.

Enter Text Boxes

The text box blocks hold descriptions about function block components. Text boxes follow this format.

```
TEXT_BOX (
       ID := <unique_identifier>,
       X := <internal_grid_x_location>,
       Y := <internal_grid_y_location>,
       Width := <numerical_value>
       Text := <"text">)
END_TEXT_BOX
```

Where:

Item	Identifies
`ID`	The text box identifier. This ID uniquely identifies this text box from all other blocks. Enter an unsigned, 32-bit integer value. Specify `ID := number`
`X`	X-coordinate on internal grid. Enter an unsigned, 32-bit integer value. Specify `X := number`
`Y`	Y-coordinate on internal grid. Enter an unsigned, 32-bit integer value. Specify `Y := number`
`Width`	This attribute is not currently used; it is there for future use. Enter 0. Specify `Width := 0`
`Text`	The descriptive text. Specify `Text := text`

TEXT_BOX Guidelines

- All TEXT_BOX blocks in function block routines must come after all BLOCK sections.
- Text boxes can be free-standing or they can be attached to FBD elements.

TEXT_BOX Example

```
TEXT_BOX (ID := 7, X := 40, Y := 80, Width := 0,
       Text := "Group these blocks together")
END_TEXT_BOX
```

Enter Attachments

The attachment blocks identify the attachments from text boxes to other function block elements. Attachments follow this format.

```
ATTACHMENT (
       FromElementID := <unique_identifier>,
       ToElementID := <unique_identifier>,
END_ATTACHMENT
```

Where:

Item	Identifies
`FromElementID`	The ID of the attached object. Enter an unsigned, 32-bit integer value. Specify `FromElementID := number`
`ToElementID`	The ID of the object that the FromID object is attached to. Enter an unsigned, 32-bit integer value. Specify `ToElementID := number`

ATTACHMENT Guidelines

- Use an attachment to link a text box to an FBD element.
- All ATTACHMENT blocks must come after all TEXT_BOX blocks.

ATTACHMENT Example

```
ATTACHMENT (FromElementID := 7, ToElementID := 2)
END_ATTACHMENT
```

Enter Parameters for Function Block Instructions

The following tables lists each function block instruction and its format in the Block component of an import/export file. For details about a specific instruction, see one of these manuals.

Instruction Type	Resource
General, sequential instruction	Logix5000 Controllers General Instructions Set Reference Manual, publication 1756-RM003
Process control or drives instruction	Logix5000 Controllers Process Control and Drives Instructions Set Reference Manual, publication 1756-RM006
Motion instruction	Logix5000 Controllers Motion Instructions Set Reference Manual, publication 1756-RM007

Instruction	Default Operand and VisiblePins formats (components within the Block structure)
ABS	Operand := ABS_01, VisiblePins := "*Source, Destination*")
ACS	Operand := ACS_01, VisiblePins := "*Source, Destination*")
ADD	Operand := ADD_01, VisiblePins := "*SourceA, SourceB, Destination*")
ALM	Operand := ALM_01, VisiblePins := "*In, HHAlarm, HAlarm, LAlarm, LLAlarm, ROCPosAlarm, ROCNegAlarm*")
ALMA	Operand := ALMA_01, VisiblePins := "*In, HHInAlarm, HInAlarm, LInAlarm, LLInAlarm, ROCPosInAlarm, ROCNegInAlarm, HHAcked, HAcked, LAcked, LLAcked, ROCPosAcked, ROCNegAcked, Suppressed, Disabled*")
ALMD	Operand := ALMD_01, VisiblePins := "*In, InAlarm, Acked, Suppressed, Disabled*")
AND	Operand := AND_01, VisiblePins := "*SourceA, SourceB, Destination*")
ASN	Operand := ASN_01, VisiblePins := "*Source, Destination*")
ATN	Operand := ATN_01, VisiblePins := "*Source, Destination*")
BAND	Operand := BAND_01, VisiblePins := "*In1, In2, In3, In4, Out*")
BNOT	Operand := BNOT_01, VisiblePins := "*In, Out*")
BOR	Operand := BOR_01, VisiblePins := "*In1, In2, In3, In4, Out*")
BTDT	Operand := BTDT_01, VisiblePins := "*Source, SourceBit, Length, DestBit, Target, Dest*")

Instruction	Default Operand and VisiblePins formats (components within the Block structure)
BXOR	Operand := BXOR_01, VisiblePins := "*In1, In2, Out*")
COS	Operand := COS_01, VisiblePins := "*Source, Dest*")
CTUD	Operand := CTUD_01, VisiblePins := "*CUEnable, CDEnable, PRE, Reset, ACC, DN*")
D2SD	Operand := D2SD_01, VisiblePins := "*ProgCommand, State0Perm, State1Perm, FB0, FB1, HandFB, ProgProgReq, ProgOperReq, ProgOverrideReq, ProgHandReq, Out, Device0State, Device1State, CommandStatus, FaultAlarm, ModeAlarm, ProgOper, Override, Hand*")
D3SD	Operand := D3SD_01, VisiblePins := "*Prog0Command, Prog1Command, Prog2Command, State0Perm, State1Perm, State2Perm, FB0, FB1, FB2, FB3, HandFB0, HandFB1, HandFB2, ProgProgReq, ProgOperReq, ProgOverrideReq, ProgHandReq, Out0, Out1, Out2, Device0State, Device1State, Device2State, Command0Status, Command1Status, Command2Status, FaultAlarm, ModeAlarm, ProgOper, Override, Hand*")
DEDT	Operand := DEDT_01, VisiblePins := "*In, Out*", Storage := *array_name*)
DEG	Operand := DEG_01, VisiblePins := "*Source, Dest*")
DERV	Operand := DERV_01, VisiblePins := "*In, ByPass, Out*")
DFF	Operand := DFF_01, VisiblePins := "*D, Clear, Clock, Q, QNot*")
DIV	Operand := DIV_01, VisiblePins := "*SourceA, SourceB, Dest*")
ESEL	Operand := ESEL_01, VisiblePins := "*In1, In2, In3, In4, In5, In6, ProgSelector, ProgProgReq, ProgOperReq, ProgOverrideReq, Out, SelectedIn, ProgOper, Override*")
EQU	Operand := EQU_01, VisiblePins := "*SourceA, SourceB*")
FGEN	Operand := FGEN_01, VisiblePins := "*In, Out*", X1 := *array_name*, X2 := *array_name*, Y2 := *array_name*, Y2 := *array_name*)
FRD	Operand := FRD_01, VisiblePins := "*Source, Dest*")

Instruction	Default Operand and VisiblePins formats (components within the Block structure)
GEQ	Operand := GEQ_01, VisiblePins := "*SourceA, SourceB*")
GRT	Operand := GRT_01, VisiblePins := "*SourceA, SourceB*")
HLL	Operand := HLL_01, VisiblePins := "*In, Out, HighAlarm, LowAlarm*")
HPF	Operand := HPF_01, VisiblePins := "*In, Out*")
INTG	Operand := INTG_01, VisiblePins := "*In, Out*")
JKFF	Operand := JKFF_01, VisiblePins := "*Clear, Clock, Q, QNot*")
LEQ	Operand := LEQ_01, VisiblePins := "*SourceA, SourceB*")
LES	Operand := LES_01, VisiblePins := "*SourceA, SourceB*")
LIM	Operand := LIM_01, VisiblePins := "*LowLlimit, Test, HighLimit*")
LN	Operand := LN_01, VisiblePins := "*Source, Dest*")
LOG	Operand := LOG_01, VisiblePins := "*Source, Dest*")
LPF	Operand := LPF_01, VisiblePins := "*In, Out*")
MAVE	Operand := MAVE_01, VisiblePins := "*In, Out*", Storage := *array_name*, Weight := *array_name*)
MAXC	Operand := MAXC_01, VisiblePins := "*In, Reset, ResetValue, Out*")
MEQ	Operand := MEQ_01, VisiblePins := "*Source, Mask, Compare*")
MINC	Operand := MINC_01, VisiblePins := "*In, Reset, ResetValue, Out*")
MOD	Operand := MOD_01, VisiblePins := "*SourceA, SourceB, Dest*")
MSTD	Operand := MSTD_01, VisiblePins := "*In, SampleEnable, Out*", Storage := *array_name*)

Instruction	Default Operand and VisiblePins formats (components within the Block structure)
MUL	Operand := MUL_01, VisiblePins := "*SourceA, SourceB, Dest*")
MUX	Operand := MUX_01, VisiblePins := "*In1, In2, In3, In4, In5, In6, In7, In8, Selector, Out*")
MVMT	Operand := MVMT_01, VisiblePins := "*Source, Mask, Target, Dest*")
NEG	Operand := NEG_01, VisiblePins := "*Source, Dest*")
NEQ	Operand := NEQ_01, VisiblePins := "*SourceA, SourceB*")
NOT	Operand := NOT_01, VisiblePins := "*Source, Dest*")
NTCH	Operand := NTCH_01, VisiblePins := "*In, Out*")
OR	Operand := OR_01, VisiblePins := "*SourceA, SourceB, Dest*")
OSFI	Operand := OSFI_01, VisiblePins := "*InputBit, OutputBit*")
OSRI	Operand := OSRI_01, VisiblePins := "*InputBit, OutputBit*")
PI	Operand := PI_01, VisiblePins := "*In, Out*")
PIDE	Operand := PIDE_01, VisiblePins := "*PV, SPProg, SPCascade, RatioProg, CVProg, FF, HandFB, ProgProgReq, ProgOperReq, ProgCasRatReq, ProgAutoReq, ProgManuaReq, ProgOverrideReq, ProgHandReq, CVEU, SP, PVHHAlarm, PVHAlarm, PVLAlarm, PVLLAlarm, PVROCPosAlarm, PVROCNegAlarm, DevHHAlarm, DevHAlarm, DevLAlarm, DevLLAlarm, ProgOper, CasRat, Auto, Manual, Override, Hand*")
PMUL	Operand := PMUL_01, VisiblePins := "*In, Multipler, Out*")
POSP	Operand := POSP_01, VisiblePins := "*SP, Position, OpenedFB, ClosedFB, OpenOut, CloseOut*")
RAD	Operand := RAD_01, VisiblePins := "*Source, Dest*")
RESD	Operand := RESD_01, VisiblePins := "*Set, Reset, Out, OutNot*")
RLIM	Operand := RLIM_01, VisiblePins := "*In, ByPass, Out*")

Instruction	Default Operand and VisiblePins formats (components within the Block structure)
RMPS	Operand := RMPS_01, VisiblePins := *"PV, CurrentSegProg, OutProg, SoakTimeProg, ProgProgReq, ProgOperReq, ProgAutoReq, ProgManualReq, ProgHoldReq, Out, CurrentSeg, SoakTimeLeft, GuarRampOn, GuarSoakOn, ProgOper, Auto, Manual, Hold"*, RampValue := *array_name*, SoakValue := *array_name*, SoakTime := *array_name*)
RTOR	Operand := RTOR_01, VisiblePins := *TimerEnable, PRE, Reset, ACC, DN")*
SCL	Operand := SCL_01, VisiblePins := "*In, Out*")
SCRV	Operand := SCRV_01, VisiblePins := "*In, Out*")
SEL	Operand := SEL_01, VisiblePins := "*In1, In2, SelectorIn, Out*")
SETD	Operand := SETD_01, VisiblePins := "*Set, Reset, Out, OutNot*")
SIN	Operand := SIN_01, VisiblePins := SIN(*source,destination*);
SNEG	Operand := SNEG_01, VisiblePins := "*In, NegateEnable, Out*")
SOC	Operand := SOC_01, VisiblePins := "*In, Out*")
SQR	Operand := SQR_01, VisiblePins := "*Source, Dest"*)
SRTP	Operand := SRTP_01, VisiblePins := "*In, HeatOut, CoolOut, HeatTimePercent, CoolTimePercent*")
SSUM	Operand := SSUM_01, VisiblePins := "*In1, Select1, In2, Select2, In3, Select3, In4, Select4, Out*")
SUB	Operand := SUB_01, VisiblePins := "*SourceA, SourceB, Dest"*)
TAN	Operand := TAN_01, VisiblePins := "*Source, Dest")*
TOD	Operand := TOD_01, VisiblePins := "*Source, Dest")*
TOFR	Operand := TOFR_01, VisiblePins := "*TimerEnable, PRE, Reset, ACC, DN*")
TONR	Operand := TONR_01, VisiblePins := "*TimerEnable, PRE, Reset, ACC, DN*")

Instruction	Default Operand and VisiblePins formats (components within the Block structure)
TOT	`Operand := TOT_01,` `VisiblePins := "In, ProgProgReq, ProgOperReq, ProgStartReq, ProgStopReq, ProgResetReq, Total, OldTotal, ProgOper, RunStop, ProgResetDone, TargetFlag, TargetDev1Flag, TargetDev2Flag")`
TRN	`Operand := TRN_01,` `VisiblePins := "Source, Dest")`
UPDN	`Operand := UPDN_01,` `VisiblePins := "InPlus, InMinus, Out")`
XOR	`Operand := XOR_01,` `VisiblePins := "SourceA, SourceB, Dest")`
XPY	`Operand := XPY_01,` `VisiblePins := "SourceA, SourceB, Dest")`

Enter Sequential Function Chart Logic

Introduction

This chapter explains how to enter sequential function chart logic in a complete import/export file.

Topic	Page
Enter a Sequential Function Chart Routine	131
Export Sequential Function Chart Logic While Editing Online	139
Enter Steps	141
Enter Transitions	145
Enter Subroutine Calls	147
Enter Stops	148
Enter Branches	149
Enter Directed Links	151
Enter Text Boxes	152
Enter Attachments	153

For more information on creating SFCs and correct syntax, see the Logix5000 Controller Common Procedures Programming Manual, publication 1756-PM001.

Enter a Sequential Function Chart Routine

Enter sequential function chart logic in an SFC_ROUTINE component in an import/export file. Each routine follows this structure.

```
SFC_ROUTINE <routine_name> [Attributes]
       <STEP_component>
       <TRANSITION_componment>
       <SBR_RET_component>
       <STOP_component>
       <BRANCH_component>
       <DIRECTED_LINK_component>
       <TEXT_BOX_component)
       <ATTACHMENT_component>
END_SFC_ROUTINE
```

Where:

Item	Identifies
`routine_name`	The name of the SFC routine.
`Attributes`	Attributes of the SFC routine.
`STEP_component`	SFC step block, contains actions.
`TRANSITION_component`	SFC transition block.
`SBR_RET_component`	Subroutine call.
`STOP_component`	SFC stop block.
`BRANCH_component`	SFC branch.
`DIRECTED_LINK_component`	SFC directed link.
`TEXT_BOX_component`	SFC text box.
`ATTACHMENT_component`	SFC attachment.

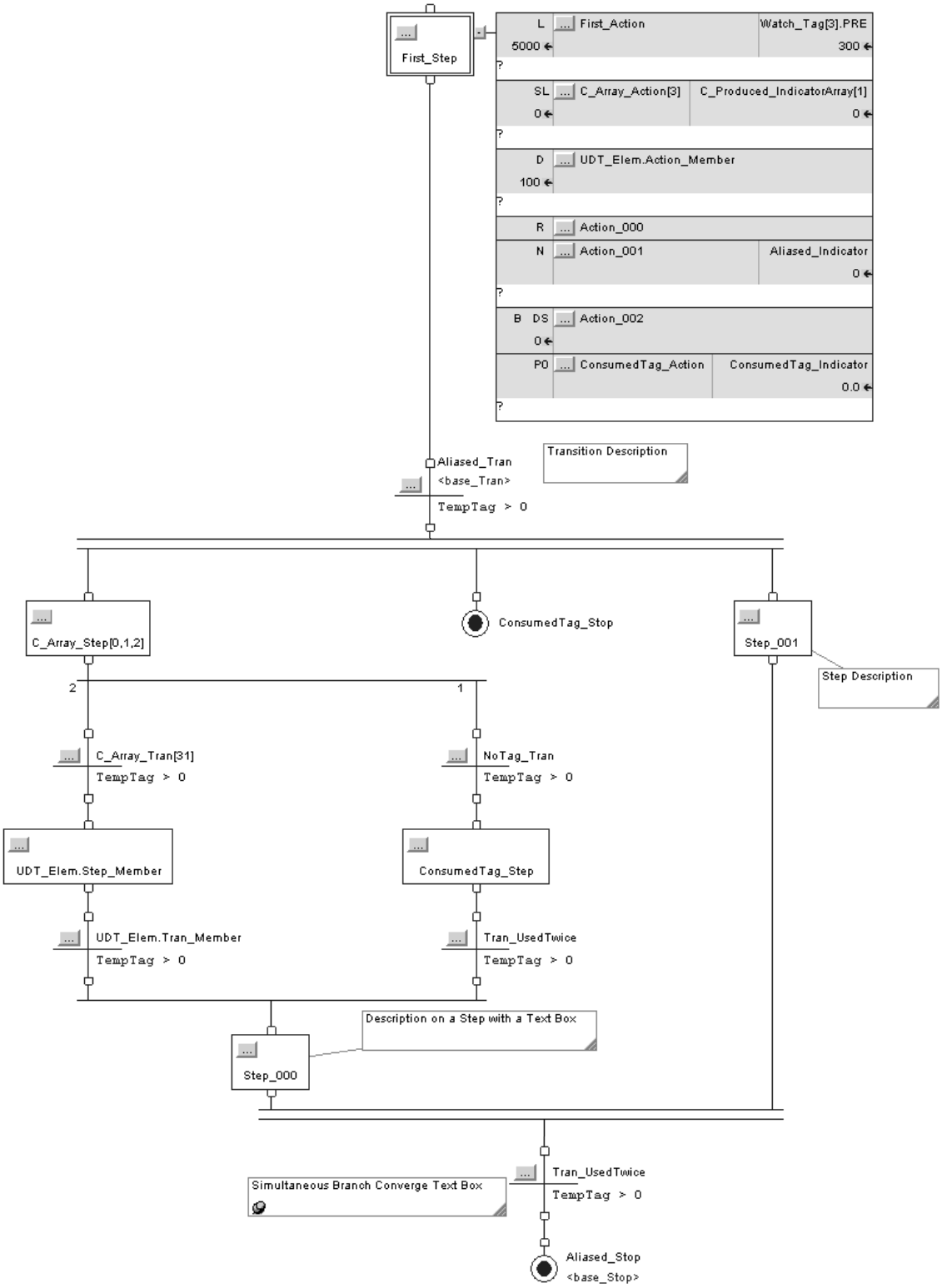
Specify SFC_ROUTINE Attributes

Where:

Item	Identifies
`Description`	Provide information about the routine. Specify `Description := "text"`
`SheetSize`	The size of the SFC. Select one of these options. • Letter (8.5x11in) • Legal (8.5x14in) • Tabloid (11x17in) • A4 (210x297mm) • A3 (297x420mm) Specify `SheetSize := option`
`SheetOrientation`	The orientation of the SFC sheet. Select Portrait or Landscape. Specify `SheetOrientation := option`
`StepName`	The prefix for the name of the step blocks within this SFC routine. RSLogix 5000 software uses this prefix when it automatically generates an SFC_STEP tag. Specify `StepName := name`
`TransitionName`	The prefix for the name of the transition blocks with this SFC routine. RSLogix 5000 software uses this prefix when it automatically generates a transition tag. Specify `TransitionName := name`
`ActionName`	The prefix for the name of the action blocks in this SFC routine. RSLogix 5000 software uses this prefix when it automatically generates an SFC_ACTION tag. Specify `ActionName := name`
`StopName`	The prefix for the name of the stop blocks in this SFC routine. RSLogix 5000 software uses this prefix when it automatically generates an SFC_STOP tag. Specify `StopName := name`

SFC_ROUTINE Example

This SFC routine:



exports to this:

```
SFC_ROUTINE Sample_SFC_Routine1 (SheetSize := "Letter (8.5x11in)",
                                 SheetOrientation := Landscape,
StepName := "Step",
                                 TransitionName := "Tran",
ActionName := "Action",
                                 StopName := "Stop")
       TRANSITION  (ID := 0, X := 120, Y := 1000, Operand :=
C_Array_Tran[31],
              HideDescription := Yes, DescriptionX := 155,
DescriptionY := 985,
              DescriptionWidth := 0)
              CONDITION  (LanguageType := ST)
              'TempTag > 0
              END_CONDITION
       END_TRANSITION
       BRANCH  (ID := 2, Y := 820, BranchType := Simultaneous,
BranchFlow := Diverge)
              LEG  (ID := 3)
              END_LEG
              LEG  (ID := 4)
              END_LEG
              LEG  (ID := 5)
              END_LEG
       END_BRANCH
       TRANSITION  (ID := 6, X := 420, Y := 760, Operand :=
Aliased_Tran,
              HideDescription := No, DescriptionX := 520,
DescriptionY := 740,
              DescriptionWidth := 0)
              CONDITION  (LanguageType := ST)
              'TempTag > 0
              END_CONDITION
       END_TRANSITION
       STOP  (ID := 8, X := 460, Y := 880, Operand :=
ConsumedTag_Stop,
              HideDescription := Yes, DescriptionX := 565,
DescriptionY := 865,
              DescriptionWidth := 0)
       END_STOP
       TRANSITION  (ID := 10, X := 520, Y := 1360, Operand :=
Tran_UsedTwice,
              HideDescription := Yes, DescriptionX := 555,
DescriptionY := 1345,
              DescriptionWidth := 0)
              CONDITION  (LanguageType := ST)
              'TempTag > 0
              END_CONDITION
       END_TRANSITION
```

```
        TRANSITION  (ID := 12, X := 460, Y := 1160, Operand :=
Tran_UsedTwice,
                HideDescription := Yes, DescriptionX := 495,
DescriptionY := 1145,
                DescriptionWidth := 0)
                CONDITION  (LanguageType := ST)
                'TempTag > 0
                END_CONDITION
        END_TRANSITION


        BRANCH  (ID := 14, Y := 940, BranchType := Selection,
BranchFlow := Diverge,
                Priority := UserDefined)
                LEG  (ID := 15)
                END_LEG
                LEG  (ID := 16)
                END_LEG
        END_BRANCH
        BRANCH  (ID := 17, Y := 1320, BranchType := Simultaneous,
BranchFlow := Converge)
                LEG  (ID := 18)
                END_LEG
                LEG  (ID := 19)
                END_LEG
        END_BRANCH
        STOP  (ID := 20, X := 520, Y := 1440, Operand := Aliased_Stop,
HideDescription := No,
                DescriptionX := 400, DescriptionY := 1480,
DescriptionWidth := 0)
        END_STOP
        STEP  (ID := 22, X := 420, Y := 360, Operand := First_Step,
HideDescription := Yes,
                DescriptionX := 478, DescriptionY := 345,
DescriptionWidth := 0,
                InitialStep := Yes, PresetUsesExpression := No,
LimitHighUsesExpression := No,
                LimitLowUsesExpression := No, ShowActions := Yes)
                ACTION  (ID := 24, Operand := First_Action, Qualifier
:= L, IsBoolean := No,
                PresetUsesExpression := No, IndicatorTag :=
Watch_Tag[3].PRE)
                BODY  (LanguageType := ST)
                '
                END_BODY
                END_ACTION
                ACTION  (ID := 25, Operand := C_Array_Action[3],
Qualifier := SL,
                IsBoolean := No, PresetUsesExpression := No,
                IndicatorTag := C_Produced_IndicatorArray[1])
                BODY  (LanguageType := ST)
```

```
                '
                END_BODY
                END_ACTION
                ACTION  (ID := 26, Operand := UDT_Elem.Action_Member,
Qualifier := D,
                IsBoolean := No, PresetUsesExpression := No,
IndicatorTag := "")
                BODY  (LanguageType := ST)
                '
                END_BODY
                END_ACTION
                ACTION  (ID := 27, Operand := Action_000, Qualifier :=
R, IsBoolean := No,
                PresetUsesExpression := No, IndicatorTag := "")
                BODY  (LanguageType := ST)
                '
                END_BODY
                END_ACTION


                ACTION  (ID := 28, Operand := Action_001, Qualifier :=
N, IsBoolean := No,
                PresetUsesExpression := No, IndicatorTag :=
Aliased_Indicator)
                BODY  (LanguageType := ST)
                '
                END_BODY
                END_ACTION
                ACTION  (ID := 29, Operand := Action_002, Qualifier :=
DS, IsBoolean := Yes,
                PresetUsesExpression := No, IndicatorTag := "")
                END_ACTION
                ACTION  (ID := 30, Operand := ConsumedTag_Action,
Qualifier := P0,
                IsBoolean := No, PresetUsesExpression := No,
                IndicatorTag := ConsumedTag_Indicator)
                BODY  (LanguageType := ST)
                '
                END_BODY
                END_ACTION
        END_STEP
        STEP  (ID := 31, X := 120, Y := 880, Operand :=
"C_Array_Step[0,1,2]",
                HideDescription := Yes, DescriptionX := 179,
DescriptionY := 865,
                DescriptionWidth := 0, InitialStep := No,
PresetUsesExpression := No,
                LimitHighUsesExpression := No, LimitLowUsesExpression
:= No, ShowActions := Yes)
        END_STEP
```

```
        TRANSITION  (ID := 33, X := 460, Y := 1000, Operand :=
NoTag_Tran,
                HideDescription := Yes, DescriptionX := 495,
DescriptionY := 985,
                DescriptionWidth := 0)
                CONDITION  (LanguageType := ST)
                'TempTag > 0
                END_CONDITION
        END_TRANSITION
        STEP  (ID := 35, X := 120, Y := 1080, Operand :=
UDT_Elem.Step_Member,
                HideDescription := Yes, DescriptionX := 199,
DescriptionY := 1065,
                DescriptionWidth := 0, InitialStep := No,
PresetUsesExpression := No,
                LimitHighUsesExpression := No, LimitLowUsesExpression
:= No, ShowActions := Yes)
        END_STEP
        STEP  (ID := 37, X := 720, Y := 880, Operand := Step_001,
HideDescription := No,
                DescriptionX := 760, DescriptionY := 940,
DescriptionWidth := 0,
                InitialStep := No, PresetUsesExpression := No,
LimitHighUsesExpression := No,
                LimitLowUsesExpression := No, ShowActions := Yes)
        END_STEP
        BRANCH  (ID := 39, Y := 1220, BranchType := Selection,
BranchFlow := Converge)
                LEG  (ID := 40)
                END_LEG
                LEG  (ID := 41)
                END_LEG
        END_BRANCH


        STEP  (ID := 42, X := 280, Y := 1260, Operand := Step_000,
HideDescription := No,
                DescriptionX := 360, DescriptionY := 1240,
DescriptionWidth := 0,
                InitialStep := No, PresetUsesExpression := No,
LimitHighUsesExpression := No,
                LimitLowUsesExpression := No, ShowActions := Yes)
        END_STEP
        STEP  (ID := 44, X := 460, Y := 1080, Operand :=
ConsumedTag_Step,
                HideDescription := Yes, DescriptionX := 514,
DescriptionY := 1065,
                DescriptionWidth := 0, InitialStep := No,
PresetUsesExpression := No,
                LimitHighUsesExpression := No, LimitLowUsesExpression
:= No, ShowActions := Yes)
        END_STEP
```

```
        TRANSITION  (ID := 46, X := 120, Y := 1160, Operand :=
UDT_Elem.Tran_Member,
                HideDescription := Yes, DescriptionX := 155,
DescriptionY := 1145,
                DescriptionWidth := 0)
                CONDITION  (LanguageType := ST)
                'TempTag > 0
                END_CONDITION
        END_TRANSITION
        DIRECTED_LINK  (FromElementID := 46, ToElementID := 41,
ShowLink := True)
        END_DIRECTED_LINK
        DIRECTED_LINK  (FromElementID := 15, ToElementID := 33,
ShowLink := True)
        END_DIRECTED_LINK
        DIRECTED_LINK  (FromElementID := 35, TToElementID := 46,
ShowLink := True)
        END_DIRECTED_LINK
        DIRECTED_LINK  (FromElementID := 3, ToElementID := 37,
ShowLink := True)
        END_DIRECTED_LINK
        DIRECTED_LINK  (FromElementID := 5, ToElementID := 31,
ShowLink := True)
        END_DIRECTED_LINK
        DIRECTED_LINK  (FromElementID := 6, ToElementID := 2, ShowLink
:= True)
        END_DIRECTED_LINK
        DIRECTED_LINK  (FromElementID := 22, ToElementID := 6,
ShowLink := True)
        END_DIRECTED_LINK
        DIRECTED_LINK  (FromElementID := 16, ToElementID := 0,
ShowLink := True)
        END_DIRECTED_LINK
        DIRECTED_LINK  (FromElementID := 44, ToElementID := 12,
ShowLink := True)
        END_DIRECTED_LINK
        DIRECTED_LINK  (FromElementID := 33, ToElementID := 44,
ShowLink := True)
        END_DIRECTED_LINK
        DIRECTED_LINK  (FromElementID := 17, ToElementID := 10,
ShowLink := True)
        END_DIRECTED_LINK
        DIRECTED_LINK  (FromElementID := 42, ToElementID := 19,
ShowLink := True)
        END_DIRECTED_LINK
        DIRECTED_LINK  (FromElementID := 37, ToElementID := 18,
ShowLink := True)
        END_DIRECTED_LINK
        DIRECTED_LINK  (FromElementID := 4, ToElementID := 8, ShowLink
:= True)
        END_DIRECTED_LINK
```

```
        DIRECTED_LINK  (FromElementID := 39, ToElementID := 42,
ShowLink := True)
        END_DIRECTED_LINK
        DIRECTED_LINK  (FromElementID := 10, ToElementID := 20,
ShowLink := True)
        END_DIRECTED_LINK
        DIRECTED_LINK  (FromElementID := 0, ToElementID := 35,
ShowLink := True)
        END_DIRECTED_LINK
        DIRECTED_LINK  (FromElementID := 31, ToElementID := 14,
ShowLink := True)
        END_DIRECTED_LINK
        DIRECTED_LINK  (FromElementID := 12, ToElementID := 40,
ShowLink := True)
        END_DIRECTED_LINK
        TEXT_BOX  (ID := 48, X := 260, Y := 1380, Width := 0,
                Text := "Simultaneous Branch Converge Text Box")
        END_TEXT_BOX
        ATTACHMENT  (FromElementID := 48, ToElementID := 17)
        END_ATTACHMENT
END_SFC_ROUTINE
```

Export Sequential Function Chart Logic While Editing Online

If you export sequential function chart logic that contains online edits, the export file exports LOGIC blocks to indicate the original, test edits, and pending edits states. If there are no online edits, you will not see LOGIC blocks.

Example 1: Both Test Edits and Pending Edits Exist

```
SFC_ROUTINE MySFCRoutine (SheetSize := "Letter (8.5x11in)",
              SheetOrientation := Landscape, StepName :=
"Step",
              TransitionName := "Tran", ActionName :=
"Action",
              StopName := "Stop")
```

```
        LOGIC (Online_Edit_Type := Orig)
              (* SFC logic here *)
        END_LOGIC

        LOGIC (Online_Edit_Type := Test)
              (* SFC logic here *)
        END_LOGIC

        LOGIC (Online_Edit_Type := Pend)
              (* SFC logic here *)
        END_LOGIC
END_SFC_ROUTINE
```

Example 2: Only Pending Edits Exist

```
SFC_ROUTINE MySFCRoutine (SheetSize := "Letter (8.5x11in)",
              SheetOrientation := Landscape, StepName :=
"Step",
              TransitionName := "Tran", ActionName :=
"Action",
              StopName := "Stop")
        LOGIC (Online_Edit_Type := Orig)
              (* SFC logic here *)
        END_LOGIC

        LOGIC (Online_Edit_Type := Pend)
              (* SFC logic here *)
        END_LOGIC
END_SFC_ROUTINE
```

Where:

Item	Identifies
`Online_Edit_Type`	Whether online edits exist when the logic is exported. If online edits exist, there will be a LOGIC block for Online_Edit_Type := Orig and then the appropriate LOGIC block for the existing edits. Online_Edit_Type : = Pend indicates pending edits. Online_Edit_Type := Test indicates test edits. If there are no online edits when the logic is exported, there are no LOGIC blocks and the main components in the routine are SFC logic components.

Enter Steps

Steps follow this format.

```
STEP (
        ID := <unique_identifier>,
        X := <internal_grid_x_location>,
        Y := <internal_grid_y_location>,
        Operand := <tag_reference>,
        HideDescription := <yes|no>
        DescriptionX := <numerical_value>,
        DescriptionY := <numerical_value>,
        DescriptionWidth := <numerical_value>,
        InitialStep := <yes|no>,
        PresetUsesExpression := <yes|no>,
        LimitHighUsesExpression := <yes|no>,
        LimitLowUsesExpression := <yes|no>,
        ShowActions := <yes|no>)
        <PRESET_block>
        <LIMIT_HIGH_block>
        <LIMIT_LOW_block>
        <ACTION_LIST_block>
END_STEP
```

Where:

Item	Identifies
`ID`	The step identifier. This ID uniquely identifies this step from all other blocks. Enter an unsigned, 32-bit integer value. Specify `ID := number`
`X`	X-coordinate on internal grid. Enter an unsigned, 32-bit integer value. Specify `X := number`
`Y`	Y-coordinate on internal grid. Enter an unsigned, 32-bit integer value. Specify `Y := number`
`Operand`	The step tag. Enter a tag of datatype SFC_STEP. The import process uses this tag name to name the step. Specify `Operand := tag`
`HideDescription`	Whether or not to hide the step description. Enter Yes or No. `Specify HideDescription := text`
`DescriptionX`	X-coordinate on internal grid of the description box. Enter an unsigned, 32-bit integer value. Specify `DescriptionX := number`
`DescriptionY`	Y-coordinate on internal grid of the description box. Enter unsigned, 32-bit integer value. Specify `DescriptionY := number`
`DescriptionWidth`	This attribute is not currently used; it is there for future use. Enter 0. Specify `DescriptionWidth := 0`

Item	Identifies
`InitialStep`	Whether this step is the initial step of the routine. Enter Yes or No. If you have multiple steps identified as the initial step (this is incorrect syntax), the import process designates the last initial step it encounters as the initial step and removes the initial step indicators from any other steps. `Specify InitialStep := text`
`PresetUsesExpression`	Whether the preset for the step timer is a structured text expression. Enter Yes if you plan to enter an expression in a PRESET block, otherwise, enter No. `Specify PresetUsesExpression := text`
`LimitHighUsesExpression`	Whether the preset for the limit high alarm is a structured text expression. Enter Yes if you plan to enter an expression in a LIMIT_HIGH block, otherwise, enter No. `Specify LimitHighUsesExpression := text`
`LimitLowUsesExpression`	Whether the preset for the limit low alarm is a structured text expression. Enter Yes if you plan to enter an expression in a LIMIT_LOW block, otherwise, enter No. `Specify LimitLowUsesExpression := text`
`ShowActions`	Whether to show or hide the step's actions. Enter Yes or No. `Specify ShowActions := text`
`PRESET_block`	A structured text expression that specifies the preset time in milliseconds for the step timer. If the PresetUsesExpression attribute (above) is Yes, enter a PRESET block.
`LIMIT_HIGH_block`	A structured text expression that specifies the preset time in milliseconds for a limit high alarm. If the LimitHighUsesExpression attribute (above) is Yes, enter a LIMIT_HIGH block.
`LIMIT_LOW_block`	A structured text expression that specifies the preset time in milliseconds for a limit low alarm. If the LimitLowUsesExpression attribute (above) is Yes, enter a LIMIT_LOW block.
`ACTION_LIST_block`	The actions in the step.

Enter a PRESET Block

The preset block contains a structured text expression that specifies the preset time in milliseconds for the step timer. Each line of structured text begins with a single quote (').

```
PRESET (LanguageType := ST)
       '<structured_text>
END_PRESET
```

Enter a LIMIT_HIGH Block

The limit high block contains a structured text expression that specifies the preset time in milliseconds for a limit high alarm. Each line of structured text begins with a single quote (‘).

```
LIMITHIGH (LanguageType := ST)
       ‘<structured_text>
END_LIMITHIGH
```

Enter a LIMIT_LOW Block

The limit low block contains a structured text expression that specifies the preset time in milliseconds for a limit low alarm. Each line of structured text begins with a single quote (‘).

```
LIMITLOW (LanguageType := ST)
       ‘<structured_text>
END_LIMITLOW
```

Enter an ACTION_LIST Block

Each step can contain multiple actions. Each action follows this format.

```
ACTION (
       ID := <unique_identifier>,
       Operand := <tag_reference>,
       Qualifier := <character(s)>,
       IsBoolean := <yes|no>,
       PresetUsesExpression := <yes|no>,
       IndicatorTag := <tag_reference>)
       <PRESET_block>
       <BODY_block>
END_ACTION
```

Where:

Item	Identifies
`ID`	The action identifier. This ID uniquely identifies this action from all other blocks. Enter an unsigned, 32-bit integer value. Specify `ID := number`
`Operand`	The action tag. Enter a tag of datatype SFC_ACTION. The import process uses this tag name to name the action. Specify `Operand := tag`
`Qualifier`	The action qualifier. **Qualifier** **Description** N non-stored R reset S stored L time limited D time delayed P pulse P1 pulse (rising edge) P0 pulse (falling edge) SL stored and time limited SD stored and time delayed DS time delayed and stored Specify `Qualifier := character(s)`
`IsBoolean`	Whether or not the action is boolean. Enter Yes or No. Specify `IsBoolean := text`
`PresetUsesExpression`	Whether the preset for the action timer is a structured text expression. Enter Yes if you plan to enter an expression in a PRESET block, otherwise, enter No. `Specify PresetUsesExpression := text`
`IndicatorTag`	The indicator tag. Enter tag. Specify `IndicatorTag := tag`

`PRESET_block`	The preset value of the action. If the PresetUsesExpression attribute (above) is Yes, enter a PRESET block.
	
	The preset block contains a structured text expression that specifies the preset time in milliseconds for the action. Each line of structured text begins with a single quote (').
	`PRESET (LanguageType := ST)`
	`'<structured_text>`
	`END_PRESET`

`BODY_block`	The structured text of the action.
	
	The body block uses structured text to define an action. It can contain multiple structured text statements. Each line of structured text begins with a single quote (').
	`BODY (LanguageType := ST)`
	`'<structured_text>`
	`END_BODY`

STEP Example

```
      STEP (ID := 16, X := 420, Y := 360, Operand := LastStep,
HideDescription := Yes,
            DescriptionX := 474, DescriptionY := 345,
DescriptionWidth := 0,
            InitialStep := No, PresetUsesExpression := No,
LimitHighUsesExpression := No,
            LimitLowUsesExpression := No, ShowActions :=
Yes)
            ACTION (ID := 18, Operand := LastAction,
Qualifier := N, IsBoolean := No,
            PresetUsesExpression := No, IndicatorTag := "")
            BODY (LanguageType := ST)
            'LastExecuted := 1;
            END_BODY
            END_ACTION
      END_STEP
```

Enter Transitions

Transitions follow this format.

```
TRANSITION (
      ID := <unique_identifier>,
      X := <internal_grid_x_location>,
      Y := <internal_grid_y_location>,
      Operand := <tag_reference>,
      HideDescription := <yes|no>,
      DescriptionX := <numerical_value>,
      DescriptionY := <numerical_value>,
      DescriptionWidth := <numerical_value>,
      Force := <TRUE|FALSE>)
      <CONDITION_block>
END_TRANSITION
```

Where:

Item	Identifies
`ID`	The transition identifier. This ID uniquely identifies this transition from all other blocks. Enter an unsigned, 32-bit integer value. Specify `ID := number`
`X`	X-coordinate on internal grid. Enter an unsigned, 32-bit integer value. Specify `X := number`
`Y`	Y-coordinate on internal grid. Enter an unsigned, 32-bit integer value. Specify `Y := number`
`Operand`	The transition tag. Enter a boolean tag. The import process uses this tag name to name the transition. Specify `Operand := tag`
`HideDescription`	Whether or not to hide the transition description. Enter Yes or No. `Specify HideDescription := text`
`DescriptionX`	X-coordinate on internal grid of the description box. Enter an unsigned, 32-bit integer value. Specify `DescriptionX := number`
`DescriptionY`	Y-coordinate on internal grid of the description box. Enter unsigned, 32-bit integer value. Specify `DescriptionY := number`
`DescriptionWidth`	This attribute is not currently used; it is there for future use. Enter 0. Specify `DescriptionWidth := 0`
`Force`	The transition is forced. Enter TRUE for forced true (set) or enter FALSE for forced false (cleared). If the transition is not forced, do not enter this attribute. Specify `Force := text`
`CONDITION_block`	The condition to evaluate for the transition.

Enter a CONDITION Block

The condition block uses a structured text expression to specify a condition to evaluate for the transition. Each line of structured text begins with a single quote (‘).

```
CONDITION (LanguageType := ST)
       ‘<structured_text>
END_CONDITION
```

TRANSITION Example

```
      TRANSITION (ID := 14, X := 420, Y := 280, Operand :=
AlwaysTrue_002,
            HideDescription := Yes, DescriptionX := 455,
DescriptionY := 265,
            DescriptionWidth := 0)
            CONDITION (LanguageType := ST)
            '1
            END_CONDITION
      END_TRANSITION
```

Enter Subroutine Calls

The subroutine calls pass values into and out of the SFC routine. Subroutine calls follow this format.

```
SBR_RET (
      ID := <unique_identifier>,
      X := <internal_grid_x_location>,
      Y := <internal_grid_y_location>,
      In := <“list”>,
      Out := <“list”>)
END_SBR_RET
```

Where:

Item	Identifies
ID	The SBR_RET identifier. This ID uniquely identifies this subroutine call from all other blocks. Enter an unsigned, 32-bit integer value. Specify `ID := number`
X	X-coordinate on internal grid. Enter an unsigned, 32-bit integer value. Specify `X := number`
Y	Y-coordinate on internal grid. Enter an unsigned, 32-bit integer value. Specify `Y := number`
In	List of values to receive from the calling routine. Enter list of tags or literal values and separate each entry by a comma (,). Enter empty quotes if there are no values to receive. Specify `In := “list”`
Out	List of values to pass to the calling routine. Enter list of tags or literal values and separate each entry by a comma (,). Enter empty quotes if there are no values to pass. Specify `Out := “list”`

SBR_RET Example

```
SBR_RET (ID := 2, X := 80, Y := 40,
             In := "Input_000, Input_001, Input_002",
             Out := "")
END_SBR_RET
```

Enter Stops

Stops follow this format.

```
STOP (
       ID := <unique_identifier>,
       X := <internal_grid_x_location>,
       Y := <internal_grid_y_location>,
       Operand := <tag_reference>,
       HideDescription := <yes|no>
       DescriptionX := <numerical_value>,
       DescriptionY := <numerical_value>,
       DescriptionWidth := <numerical_value>)
END_STOP
```

Where:

Item	Identifies
`ID`	The stop identifier. This ID uniquely identifies this stop from all other blocks. Enter an unsigned, 32-bit integer value. Specify `ID := number`
`X`	X-coordinate on internal grid. Enter an unsigned, 32-bit integer value. Specify `X := number`
`Y`	Y-coordinate on internal grid. Enter an unsigned, 32-bit integer value. Specify `Y := number`
`Operand`	The stop tag. Enter a tag of datatype SFC_STOP. The import process uses this tag name to name the stop. Specify `Operand := tag`
`HideDescription`	Whether or not to hide the stop description. Enter Yes or No. `Specify HideDescription := text`
`DescriptionX`	X-coordinate on internal grid of the description box. Enter an unsigned, 32-bit integer value. Specify `DescriptionX := number`
`DescriptionY`	Y-coordinate on internal grid of the description box. Enter unsigned, 32-bit integer value. Specify `DescriptionY := number`
`DescriptionWidth`	This attribute is not currently used; it is there for future use. Enter 0. Specify `DescriptionWidth := 0`

STOP Example

```
       STOP (ID := 10, X := 420, Y := 520, Operand :=
NeverGetsHere, HideDescription := Yes,
              DescriptionX := 505, DescriptionY := 505,
DescriptionWidth := 0)
       END_STOP
```

Enter Branches

The branch blocks in an SFC routine identify simultaneous or selection branches in the routine.

```
BRANCH (
       ID := <unique_identifier>,
       Y := <internal_grid_y_location>,
       BranchType := <text>,
       BranchFlow := <text>,
       Priority := <text>)
       <LEG_block>
END_BRANCH
```

Where:

Item	Identifies
`ID`	The branch identifier. This ID uniquely identifies this branch from all other blocks. Enter an unsigned, 32-bit integer value. Specify `ID := number`
`Y`	Y-coordinate on internal grid. Enter an unsigned, 32-bit integer value. Specify `Y := number`
`BranchType`	The type of branch. Enter Simultaneous or Selection. Specify `BranchType := text`
`BranchFlow`	The direction of the branch. Enter Converge or Diverge. `Specify BranchFlow := text`
`Priority`	Whether the priority of a divergent selection branch is defined by the user. This attribute applies only to divergent selection branches. Enter Default or UserDefined. Specify `Priority := text`
`LEG_block`	The individual legs of the branch. Enter one leg block for each leg of the branch.

Entering the LEG Block

The leg block identifies a leg of a branch. Legs follow this format.

```
LEG (
       ID := <unique_identifier>,
       Force := <FALSE>)
END_LEG
```

Where:

Item	Identifies
`ID`	The leg identifier. This ID uniquely identifies this leg from all other blocks. Enter an unsigned, 32-bit integer value. Specify `ID := number`
`Force`	Whether the leg is forced or not. You can force only a leg in a simultaneous branch. Either omit this attribute (for no forces) or enter FALSE to force the leg false. Specify `Force := text`

BRANCH Example

```
BRANCH (ID := 4, Y := 200, BranchType := Simultaneous,
              BranchFlow := Diverge)
       LEG (ID := 5)
       END_LEG

       LEG (ID := 6)
       END_LEG

       LEG (ID := 7)
       END_LEG

END_BRANCH
```

Enter Directed Links

The directed link blocks in an SFC routine identify the links between SFC components.

```
DIRECTED_LINK (
       FromElementID := <unique_identifier>,
       ToElementID := <unique_identifier>,
       ShowLink := <TRUE|FALSE>)
END_DIRECTED_LINK
```

Where:

Item	Identifies
`FromElementID`	The source element of the link. Enter an unsigned, 32-bit integer value. Specify `FromElementID := number`
`ToElementID`	The destination element of the link. Enter an unsigned, 32-bit integer value. Specify `ToElementID := number`
`ShowLink`	Whether or not to show the link. Enter TRUE or FALSE. `Specify ShowLink := text`

DIRECTED_LINK Guidelines

- All DIRECTED_LINK blocks must come after all STEP, TRANSITION, STOP, and BRANCH blocks.
- A directed link links only one element to one other element.

DIRECTED_LINK Example

```
       DIRECTED_LINK (FromElementID := 16, ToElementID := 12,
ShowLink := True)
       END_DIRECTED_LINK
```

Enter Text Boxes

The text box blocks in an SFC routine hold descriptions about SFC components. Text boxes follow this format.

```
TEXT_BOX (
       ID := <unique_identifier>,
       X := <internal_grid_x_location>,
       Y := <internal_grid_y_location>,
       Width := <numerical_value>
       Text := <"text">)
END_TEXT_BOX
```

Where:

Item	Identifies
`ID`	The text box identifier. This ID uniquely identifies this text box from all other blocks. Enter an unsigned, 32-bit integer value. Specify `ID := number`
`X`	X-coordinate on internal grid. Enter an unsigned, 32-bit integer value. Specify `X := number`
`Y`	Y-coordinate on internal grid. Enter an unsigned, 32-bit integer value. Specify `Y := number`
`Width`	This attribute is not currently used; it is there for future use. Enter 0. Specify `Width := 0`
`Text`	The descriptive text. Specify `Text := text`

TEXT_BOX Guidelines

- All TEXT_BOX blocks must come after all DIRECTED_LINK blocks.
- Text boxes can be free-standing or they can be attached to SFC elements.

TEXT_BOX Example

```
TEXT_BOX (ID := 7, X := 40, Y := 80, Width := 0,
       Text := "Action Body makes recursive call")
END_TEXT_BOX
```

Enter Attachments

The attachment blocks in an SFC routine identify the attachments from text boxes to other SFC elements.

```
ATTACHMENT (
        FromElementID := <unique_identifier>,
        ToElementID := <unique_identifier>,
END_ATTACHMENT
```

Where:

Item	Identifies
`FromElementID`	The ID of the attached object. Enter an unsigned, 32-bit integer value. Specify `FromElementID := number`
`ToElementID`	The ID of the object that the FromID object is attached to. Enter an unsigned, 32-bit integer value. Specify `ToElementID := number`

ATTACHMENT Guidelines

- Use an attachment to link a text box to an SFC element.
- All ATTACHMENT blocks must come after all TEXT_BOX blocks.

ATTACHMENT Example

```
ATTACHMENT (FromElementID := 7, ToElementID := 2)
END_ATTACHMENT
```

Notes:

Enter Structured Text Logic

Introduction

This chapter explains the how to enter structured text logic in a complete import/export file.

Topic	Page
Enter a Structured Text Routine	155
Enter Structured Text Logic	156
Enter Comments	157
Export Structured Text Logic While Editing Online	158
Enter Structured Text	159

Enter a Structured Text Routine

A structured text ST_ROUTINE follows this structure.

```
ST_ROUTINE <routine_name> [Attributes]

       '<statements>;

END_ST_ROUTINE;
```

Where:

Item	Identifies
`<routine_name>`	the name of the structured text routine
`Attributes`	attributes of the structured text routine
`<statements>`	structured text logic every line must begin with a single quote (')

Specify ST_ROUTINE attributes

Specify these attributes for a ROUTINE.

Attribute	Description
`Description`	Provide information about the routine. Specify `Description := "text"`

Enter Structured Text Logic

Enter structured text logic within an ST_ROUTINE component in an import/export file. Each line of structured text must begin with a single quote (').

Structured text is not case sensitive. Structured text can contain these elements.

Term	Definition		Examples
Assignment	Use an assignment statement to assign values to tags. The := operator is the assignment operator. Terminate the assignment with a semi colon ";".		`tag := expression;`
Expression	An expression is part of a complete assignment or construct statement. An expression evaluates to a number (numerical expression) or to a true or false state (BOOL expression). An expression contains these elements.		
	Tags	A named area of the memory where data is stored (BOOL, SINT,INT,DINT, REAL, string).	`value1`
	Immediates	A constant value.	`4`
	Operators	A symbol or mnemonic that specifies an operation within an expression.	`tag1 + tag2` `tag1 >= value1`
	Functions	When executed, a function yields one value. Use parentheses to contain the operand of a function. Functions can be used only in expressions.	`function(tag1)`

Instruction	An instruction is a standalone statement.
	An instruction uses parenthesis to contain its operands.
	Depending on the instruction, there can be zero, one, or multiple operands.
	When executed, an instruction yields one or more values that are part of a data structure.
	Terminate the instruction with a semi colon ";".
	
	Instructions cannot be used in expressions.
			`instruction();`
			
			`instruction(operand);`
			
			`instruction(operand1, operand2,operand3);`

Construct	A conditional statement used to trigger structured text code (i.e, other statements).
	Terminate the construct with a semi colon ";".
			`IF...THEN`
			`CASE`
			`FOR...DO`
			`WHILE...DO`
			`REPEAT...UNTIL`
			`EXIT`

Comment	Text that explains or clarifies what a section of structured text does.
	• Use comments to make it easier to interpret the structured text.
	• Comments do not affect the execution of the structured text.
	• Comments can appear anywhere in structured text.
			`//comment`
			
			`(*start of comment . . . end of comment*)`
			
			`/*start of comment . . . end of comment*/`

For details on these components, see the structured text appendix that is in both the Logix5000 Controllers General Instructions Reference Manual, publication 1756-RM003 and in the Logix5000 Controllers Process Control and Drives Instructions Reference Manual, publication 1756-RM006.

Structured Text ST_ROUTINE Example

This is an example of an exported structured text routine.

```
ST_ROUTINE <routine_name>
(*-------------------------------------------------------------------------------------------
----------- Sample of ST code ---------------------------------------------------------------
-------------------------------------------------------------------------------------------*)
             'IF (myInteger = 12) THEN
             '      myInteger := ((5 * myInputInteger1) + (7 * myInteger2)) - 71;
             '             WHILE (myTmpVar >= 0) DO
             '                    myInteger := myInteger + 3;
             '                    myTmpVar := myTmpVar - 1;
             '             END_WHILE;
             'END_IF;
END_ST_ROUTINE
```

Enter Comments

Enclose comments between (* and *) characters. Comments can include carriage returns. You can place comments anywhere in structured text logic. For example:

```
(*-------------------------------------------------------------------------------------------
----------- Example comment -----------------------------------------------------------------
-------------------------------------------------------------------------------------------*)
```

Export Structured Text Logic While Editing Online

If you export structured text logic that contains online edits, the export file exports LOGIC blocks to indicate the original, test edits, and pending edits states. If there are no online edits, you will not see these LOGIC blocks.

Example 1: Both Test Edits and Pending Edits Exist

```
ST_ROUTINE MySTRoutine
       LOGIC (Online_Edit_Type := Orig)
              (* structured text logic here *)
       END_LOGIC

       LOGIC (Online_Edit_Type := Test)
              (* structured text logic here *)
       END_LOGIC

       LOGIC (Online_Edit_Type := Pend)
              (* structured text logix here *)
       END_LOGIC
END_ST_ROUTINE
```

Example 2: Only Pending Edits Exist

```
ST_ROUTINE MySTRoutine
       LOGIC (Online_Edit_Type := Orig)
              (* structured text logic here *)
       END_LOGIC

       LOGIC (Online_Edit_Type := Pend)
              (* structured text logic here *)
       END_LOGIC
END_ST_ROUTINE
```

Where:

Item	Identifies
`Online_Edit_Type`	Whether online edits exist when the logic is exported. If online edits exist, there will be a LOGIC block for Online_Edit_Type := Orig and then the appropriate LOGIC block for the existing edits. Online_Edit_Type : = Pend indicates pending edits. Online_Edit_Type := Test indicates test edits. If there are no online edits when the logic is exported, there are no LOGIC blocks and the main components in the routine are structured text statements.

Enter Structured Text

The following tables lists each structured text instruction and function. For more details, see one of these manuals.

Instruction Type	Resource
General, sequential instruction	Logix5000 Controllers General Instructions Set Reference Manual, publication 1756-RM003
Process control or drives instruction	Logix5000 Controllers Process Control and Drives Instructions Set Reference Manual, publication 1756-RM006
Motion instruction	Logix5000 Controllers Motion Instructions Set Reference Manual, publication 1756-RM007

Instruction	Neutral Text Format
ABL	`ABL(Channel,SerialPortControl);`
ABS	`dest := ABS(source);`
ACB	`ACB(Channel,SerialPortControl);`
ACL	`ACL(Channel,ClearSerialPortRead,ClearSerialPortWrite);`
ACOS	`dest := ACOS(source);`
ADD	`dest := sourceA + sourceB;`
AHL	`AHL(Channel,ANDMask,ORMask,SerialPortControl);`
ALM	`ALM(ALM_tag);`
ALMA	`ALMA (ALMA_tag,In,ProgAckAll,ProgramDisable,ProgEnable);`
ALMD	`ALMD (ALMD_tag,In,ProgAck,ProgReset,ProgDisable,ProgEnable);`
AND	`dest := sourceA & sourceB;` `dest := sourceA AND sourceB;`
ARD	`ARD(Channel,Destination,SerialPortControl);`
ARL	`ARL(Channel,Destination,SerialPortControl);`
ASIN	`dest := ASIN(source);`
ATAN	`dest := ATAN(source);`
AWA	`AWA(Channel,Source,SerialPortControl);`
AWT	`AWT(Channel,Source,SerialPortControl);`

BAND	`IF operandA AND operandB THEN`
	`  <statement>;`
	`ENDIF;`

BNOT	`IF NOT operand THEN`
	`  <statements>;`
	`ENDIF;`

BOR	`IF operandA OR operandB THEN`
	`  <statements>;`
	`ENDIF;`

BTDT	`BTD(BTDT_tag);`

Instruction	Neutral Text Format

BXOR	`IF operandA XOR operandB THEN`
	`  <statements>;`
	`ENDIF;`

CASE...OF	`CASE numeric_expression OF`
	`  selector1: statement;`
	`  selectorN: statement;`
	`ELSE`
	`  statement;`
	`END_CASE;`

CLR	`dest := 0;`
CONCAT	`CONCAT(SourceA,SourceB,Dest)`
COP	`COP(Source,Dest,Length);`
COS	`dest := COS(source);`
CPS	`CPS(Source,Dest,Length)`
CTUD	`CTUD(CTUD_tag);`
D2SD	`D2SD(D2SD_tag);`
D3SD	`D3SD(D3SD_tag);`
DEDT	`DEDT(DEDT_tag,storage);`
DEG	`dest := DEG(source);`
DELETE	`DELETE(Source,Qty,Start,Dest);`
DERV	`DERV(DERV_tag);`
DFF	`DFF(DFF_tag);`
DIV	`dest := sourceA / sourceB;`
DTOS	`DTOS(Source,Dest);`
EOT	`EOT(DataBit);`

EQU	`IF sourceA = sourceB THEN`
	`  <statements>;`
	`ENDIF;`

ESEL	`ESEL(ESEL_tag);`
EVENT	`EVENT(task);`
FGEN	`FGEN(FGEN_tag,X1,Y1,X2,Y2);`
FIND	`FIND(Source,Search,Start,Result)`

FOR...DO	`FOR count:= initial_value TO final_value BY increment DO`
	`  <statement>;`
	`END_FOR;`

GEQ	`IF sourceA >= sourceB THEN`
	`  <statements>;`
	`ENDIF;`

Instruction	Neutral Text Format

GRT	IF *sourceA* > source*B* THEN
	  <statements>;
	ENDIF;

GSV	GSV(*ClassName,InstanceName,AttributeName,Dest*);
HLL	HLL(*HLL_tag*);
HPF	HPF(*HPF_tag*);
IF...THEN	IF *bool_expression* THEN *<statement>*; END_IF;
INSERT	INSERT(*SourceA,SourceB,Start,Dest*);
INTG	INTG(*INTG_tag*);
IOT	IOT(*output_tag*);
JKFF	JKFF(*JKFF_tag*);
JSR	JSR(*RoutineName,InputCount,InputPar,ReturnPar*);
LDL2	LDL2(*LDL2_tag*);
LDLG	LDLG(*LDLG_tag*);

LEQ	IF *sourceA* <= source*B* THEN
	  <statements>;
	ENDIF;

LES	IF *sourceA* < source*B* THEN
	  <statements>;
	ENDIF;

LN	*dest* := LN(*source*);
LOG	*dest* := LOG(*source*);
LOWER	LOWER(*Source,Dest);*
LPF	LPF(*LPF_tag*);
MAAT	MAAT(*Axis,MotionControl*);
MAFR	MAFR(*Axis,MotionControl*);
MAG	MAG(Slave*Axis,MasterAxis,MotionControl,Direction,Ratio,SlaveCounts,MasterCounts,MasterReference,RatioFormat,Clutch,AccelRate,AccelUnits*);
MAH	MAH(*Axis,MotionControl*);
MAHD	MAHD(*Axis,MotionControl,DiagnosticTest,ObservedDirection*);
MAJ	MAJ(*Axis,MotionControl,Direction,Speed,SpeedUnits,AccelRate,AccelUnits,DecelRate,DecelUnits,Profile,Merge,MergeSpeed*);
MAM	MAM(*Axis,MotionControl,MoveType,Position,Speed,SpeedUnits,AccelRate,AccelUnits,DecelRate,DecelUnits,Profile,Merge,MergeSpeed*);
MAOC	MAOC(*Axis,ExecutionTarget,MotionControl,Output,Input,OutputCam,CamStartPosition,CamEndPosition,OutputCompensation,ExecutionMode,ExecutionSchedule,AxisArmPosition,CamArmPosition,Reference*);

Instruction	Neutral Text Format
MAPC	MAPC(*SlaveAxis*,*MasterAxis*,*MotionControl*,*Direction*,*CamProfile*, *SlaveScaling*,*MasterScaling*,*ExecutionMode*,*ExecutionSchedule*, *MasterLockPosition*,*CamLockPosition*,*MasterReference*,*MasterDirection*);
MAR	MAR(*Axis,MotionControl,TriggerCondition,WindowedRegistration, MinimumPosition,MaximumPosition*);
MAS	MAS(*Axis,MotionControl,StopType,ChangeDecel,DecelRate,DecelUnits*);
MASD	MASD(*Axis,MotionControl*);
MASR	MASR(*Axis,MotionControl*);
MATC	MATC(*Axis*,*MotionControl*,*Direction*,*CamProfile*,*DistanceScaling*, *TimeScaling*,*ExecutionMode*,*ExecutionSchedule*);
MAVE	MAVE(*MAVE_tag,storage,weight*);
MAW	MAW(*Axis,MotionControl,TriggerCondition,Position*);
MAXC	MAXC(*MAXC_tag*);
MCCD	MCCD(*Coordinate_system,MotionControl,MotionType,ChangeSpeed,Speed, SpeedUnits,ChangeAccel,AccelRate,AccelUnits,ChangeDecel,DecelRate, DecelUnits,Scope*);
MCCM	MCCM(*CoordinateSystem,MotionControl,MoveType,Position,CircleType, Via/Center/Radius,Direction,Speed,SpeedUnits,AccelRate,AccelUnits, DecelRate,DecelUnits,Profile,TerminationType,Merge,MergeSpeed*);
MCCP	MCCP(*MotionControl*,*Cam*,*Length*,*StartSlope*,*EndSlope*,*CamProfile*);
MCD	MCD(A*xis,MotionControl,MotionType,ChangeSpeed,Speed,ChangeAccel, AccelRate,ChangeDecel,DecelRate,SpeedUnits,AccelUnits,DecelUnits*);
MCLM	MCLM(*CoordinateSystem,MotionControl,MoveType,Position,Speed,SpeedUnits, AccelRate,AccelUnits,DecelRate,DecelUnits,Profile,TerminationType,Merge, MergeSpeed*);
MCS	MCS(*CoordinateSystem*,*MotionControl*,*StopType*,*ChangeDecel*,*DecelRate*, *DecelUnits*);
MCSD	MCSD(*CoordinateSystem,MotionControl*);
MCSR	MCSR(*CoordinateSystem,MotionControl*);
MCSV	MCSV(*MotionControl,CamProfile,MasterValue,SlaveValue,SlopeValue, SlopeDerivative*);
MCT	MCT(*SourceSystem,TargetSystem,MotionControl,Orientation,Translation*);
MCTP	MCTP(*SourceSystem,TargetSystem,MotionControl,Orientation,Translation, TransformDirection*,*ReferencePosition*,*TransformPosition*);
MDF	MDF(*Axis,MotionControl*);
MDO	MDO(*Axis,MotionControl,DriveOutput,DriveUnits*);
MDOC	MDOC(*Axis,ExecutionTarget,MotionControl,DisarmType*);
MDR	MDR(*Axis,MotionControl*);
MDW	MDW(*Axis,MotionControl*);
MEQ	IF (Source AND Mask) = (Compare AND Mask) THEN <*statements*>; END_IF;
MGS	MGS(*Group,MotionControl,StopMode*);

Instruction	Neutral Text Format
MGSD	`MGSD(Group,MotionControl);`
MGSP	`MGSP(Group,MotionControl);`
MGSR	`MGSR(Group,MotionControl);`
MID	`MID(Source,Qty,Start,Dest);`
MINC	`MINC(MINC_tag);`
MOD	`dest := sourceA MOD sourceB;`
MRAT	`MRAT(Axis,MotionControl);`
MRHD	`MRHD(Axis,MotionControl,DiagnosticTest);`
MRP	`MRP(Axis,MotionControl,Type,PositionSelect,Position);`
MSF	`MSF(Axis,MotionControl);`
MSG	`MSG(MessageControl);`
MSO	`MSO(Axis,MotionControl);`
MUL	`dest := sourceA * sourceB;`
MVMT	`MVMT(MVMT_tag);`
NEG	`dest := -source;`

NEQ	`IF sourceA <> sourceB THEN`
	`  <statements>;`
	`END_IF;`

NOT	`IF NOT source THEN`
	`  <statements>;`
	`END_IF;`

OR	`dest := sourceA OR sourceB`
OSFI	`OSFI(OSFI_tag);`
OSRI	`OSRI(OSRI_tag);`
OTE	`data_bit [:=] BOOL_expression;`

OTL	`IF BOOL_expression THEN`
	`  data_bit := 1;`
	`END_IF;`

OTU	`IF BOOL_expression THEN`
	`  data_bit := 0;`
	`END_IF;`

PATT	`PATT(PhaseName,Result);`
PCLF	`PCLF(PhaseName);`
PCMD	`PCMD(PhaseName,Command,Result);`
PDET	`PDET(PhaseName);`
PFL	`PFL(Source);`
PI	`PI(PI_tag);`
PID	`PID(PID,ProcessVariable,Tieback,ControlVariable,PIDMasterLoop,InholdBit,InholdValue);`
PIDE	`PIDE(PIDE_tag);`

Instruction	Neutral Text Format
PMUL	`PMUL(PMUL_tag);`
POSP	`POSP(POSP_tag);`
POVR	`POVR(PhaseName,Command,Result);`
PPD	`PPD();`
PRNP	`PRNP();`
PSC	`PSC();`
PXRQ	`PXRQ(PhaseInstruction,ExternalRequest,DataValue);`
RAD	`dest := RAD(source);`

REPEAT...UNTIL	`REPEAT`
	`  <statement>;`
	`UNTIL bool_expression`
	`END_REPEAT;`

RESD	`RESD(RESD_tag);`
RET	`RET(ReturnPar);`
RLIM	`RLIM(RLIM_tag);`
RMPS	`RMPS(RMPS_tag,RampValue,SoakValue,SoakTime);`
RTOR	`RTOR(RTOR_tag);`
RTOS	`RTOS(Source,Dest)`
SBR	`SBR(InputPar);`
SCRV	`SCRV(SCRV_tag);`
SETD	`SETD(SETD_tag);`
SFP	`SFP(SFCRoutineName,TargetState);`
SFR	`SFR(SFCRoutineName,StepName);`
SIN	`dest := SIN(source);`
SIZE	`SIZE(Souce,Dimensiontovary,Size);`
SNEG	`SNEG(SNEG_tag);`
SOC	`SOC(SOC_tag);`
SQRT	`dest := SQRT(source);`
SRT	`SRT(Array,Dimtovary,Control);`
SRTP	`SRTP(SRTP_tag);`
SSUM	`SSUM(SSUM_tag);`
SSV	`SSV(ClassName,InstanceName,AttributeName,Source);`
STOD	`STOD(Source,Dest)`
STOR	`STOR(Source,Dest)`
SUB	`dest := sourceA - sourceB;`
SWPB	`SWPB(Source,OrderMode,Dest);`
TAN	`dest := TAN(source);`

Instruction	Neutral Text Format
TOFR	`TOFR(TOFR_tag);`
TONR	`TONR(TONR_tag);`
TOT	`TOT(TOT_tag);`
TRUNC	`dest := TRUNC(source);`
UID	`UID();`
UIE	`UIE();`
UPDN	`UPDN(UPDN_tag);`
UPPER	`UPPER(Source,Destination);`

WHILE...DO	`WHILE bool_expression DO`
	`  <statement>;`
	`END_WHILE;`

XIC	`IF data_bit THEN`
	`  <statement>;`
	`END_IF;`

XIO	`IF NOT data_bit THEN`
	`  <statement>;`
	`END_IF;`

XOR	`dest := sourceA XOR sourceB;`
XPY	`dest := sourceX XPY sourceY;`

Notes:

Define a TASK Component

Introduction

This chapter explains the overall structure of the TASK component.

Topic	Page
Define a TASK	167
Specify TASK Attributes	168
TASK Guidelines	168
TASK Example	169

Define a TASK

A TASK component follows this structure.

```
TASK <task_name> [(Attributes)]
       <program_name>;
END_TASK
```

Where:

Item	Identifies
`task_name`	The task.
`Attributes`	Attributes of the task. Can also contain a description of the task. Enclose in parenthesis. Separate each attribute with a comma (,).
`program_name`	Each program within the task. All program names are followed by a semi colon (;).

The maximum number of tasks depends on the type of controller.

Controller	Maximum Number of Tasks
ControlLogix	32
SoftLogix5800	32
FlexLogix	8
CompactLogix • 1768-L43 • 1769-L35CR, -L35E • 1769-L32C, -L32E • 1769-L31	 • 16 • 8 • 6 • 4
DriveLogix	8

Specify TASK Attributes

Specify these attributes for a TASK.

Attribute	Description
`Description`	Provide information about the task. Specify `Description := "text"`
`Type`	Specify the type of task (CONTINUOUS, PERIODIC, or EVENT). There can be only one continuous task. Specify `Type := type`
`Class`	Specify the class of the task. This attribute applies only to safety controller projects. Enter Standard or Safety. Specify `Class := text`
`Rate`	If the task is a periodic task, specify how often to run the task (1.000...2,000,000.000 us). Specify `Rate := number`
`Priority`	Specify the priority of a periodic task (1...15) Specify `Priority := number`
`Watchdog`	Enter the watchdog timeout for the task (1.000...2,000,000.000 us). Specify `Watchdog := number`
`EventTrigger`	Only used for event tasks. Specify the trigger for the event task. Enter Axis Home, Axis Watch, Axis Registration 1, Axis Registration 2, Motion Group Execution, EVENT Instruction Only, Module Input Data State Change, Consumed Tag, or Windows Event. Specify `EventTrigger := text`
`EventTag`	Only used for event tasks with a Consumed Tag trigger or a Module Input Data State Change trigger. Specify the tag to consume. Specify `EventTag := tag_name`
`EnableTimeout`	Enter Yes to enable timeouts for the task. Otherwise enter No. Specify `EnableTimeout := text`
`DisableUpdateOutputs`	Enter Yes to disable updates to outputs while the task executes. Otherwise enter No. The default for a periodic or continuous task is No. The default for an event task is yes. Specify `DisableUpdateOutputs := text`
`InihibitTask`	Enter Yes to inhibit the task. Otherwise enter No. Specify `InhibitTask := text`

TASK Guidelines

Keep these guidelines in mind when defining a task.

- Tasks must be defined after programs and before controller objects.
- There can be at most 32 tasks.
- There can be only one continuous task.
- A program can be scheduled under only one task.
- Scheduled programs must be defined (must exist).

TASK Example

```
TASK joe (Type := Periodic,  Priority := 8, Rate := 10000)
       sue;
       betty;
END_TASK
```

The task attributes (Type, Priority, Rate, and Watchdog) can be defined in any order. The list of programs scheduled for a task are listed in the task declarations block, as shown above. The programs are executed in the order they are specified.

Safety TASK Example

```
TASK SafetyTask (Type := PERIODIC,
       Class := Safety,
       Rate := 10,
       Priority := 10,
       Watchdog := 10,
       DisableUpdateOutputs := No,
       InhibitTask := No)
       SafetyProgram;
END_TASK
```

Notes:

Define a TREND Component

Introduction

This chapter explains the overall structure of the TREND component.

Topic	Page
Define a TREND	171
Specify TREND Attributes	172
Specify a PEN Declaration	176
TREND Guidelines	177
TREND Example	178

Define a TREND

A TREND component defines controller trend object and follows this structure.

```
TREND <trend_name> [(Attributes)]
       [Template]
       [PEN declaration]
END_TREND
```

Where:

Item	Identifies
`trend_name`	The trend.
`Attributes`	Attributes of the trend. Can also contain a description of the trend. Enclose in parenthesis. Separate each attribute with a comma (,).
`Template`	The trend template in a byte value list.
`PEN declaration`	Individual pens within the trend. Each trend can support as many as 8 pens.

Trend objects are optional. You can have as many as 32 trends per import/export file.

Specify TREND Attributes

Specify these attributes for a TREND.

Attribute	Description
`Description`	Provide information about the trend. Specify `Description := "text"`
`SamplePeriod`	Specify how often trending tags are collected in msec (1 msec...30 minutes). Specify `SamplePeriod := number`
`NumberOfCaptures`	Specifies the maximum number of captures allowed (1...100). Specify `NumberOfCaptures := number`
`CaptureSizeType`	Define how the capture size is specified. Enter Samples, TimePeriod, or NoLimit. Specify `CaptureSizeType := text`
`CaptureSize`	Specify the number of samples for each capture. The maximum number of samples is 2-hours worth of data samples or 1000 samples, whichever is greater. If the CaptureSizeType is Samples, the range is 1...(2 hours/SamplePeriod) or 1000 samples, whichever is greater. If the CaptureSizeType is TimePeriod, the range is SamplePeriod...2 hours or (SamplePeriod * 1000), whichever is greater. Specify `CaptureSize := number`
`StartTriggerType`	Specify the type of the start trigger. Enter NoTrigger or EventTrigger. Specify `StartTriggerType := text`
`StartTriggerTag1`	Specify the tag name of the first start trigger. The name must be one of the pen names. Specify `StartTriggerTag1 := text`
`StartTrigger Operation1`	Specify the operation that is applied on StartTriggerTag1, and StartTriggerTargetValue1 or StartTriggerTargetTag1. **Enter** **For** 0 Exact Equal (Tag EQU Target) 1 Trigger Level Equal (Tag = Target) 2 Not Equal (Tag != Target) 3 Less Than (Tag < Target) 4 Greater Than (Tag > Target) 5 Less Than or Equal To (Tag <= Target) 6 Greater Than or Equal To (Tag >= Target) 7 Positive Slope (slope of Tag is positive) 8 Negative Slope (slope of Tag is negative) 9 Bitwise OR ((Tag OR Target) = 0) 10 Bitwise OR ((Tag OR Target) != 0) 11 Bitwise AND ((Tag AND Target) = 0) 12 Bitwise AND ((Tag AND Target) != 0) 13 Bitwise XOR ((Tag XOR Target) = 0) 14 Bitwise XOR ((Tag XOR Target) != 0) Specify `StartTriggerOperation1 := number`
`StartTriggerTarget Type1`	Specify the type of the first start trigger target. Enter TargetValue or TargetTag. If you enter TargetValue, StartTriggerTargetValue1 is expected. Otherwise, StartTriggerTargetTag1 is expected. Specify `StartTriggerTargetType1 := text`
`StartTriggerTarget Value1`	Specify a target value if the StartTriggerTargetType1 is TargetValue. Enter a binary, octal, decimal, or hexadecimal integer number or enter a floating point number. Specify `StartTriggerTargetValue1 := text`
`StartTriggerTarget Tag1`	Specify a target tag if the StartTriggerTargetType is TargetTag. The tag must be one of the pen names. Specify `StartTriggerTargetTag1 := text`

Attribute	Description
`StartTriggerLogical Operation`	Specify a logical operation (AND or OR) that is performed on StartTriggerxxx1 and StartTriggerxxx2. StartTriggerxxx1 consists of StartTriggerTag1, StartTriggerOperation1, StartTriggerTargetType1, and StartTriggerTargetValue1 or StartTriggerTargetTag1. StartTriggerxxx2 consists of StartTriggerTag2, StartTriggerOperation2, StartTriggerTargetType2, and StartTriggerTargetValue2 or StartTriggerTargetTag2. Specify `StartTriggerLogicalOperation := text`
`StartTriggerTag2`	Specify the tag name of the second start trigger. The name must be one of the pen names. Specify `StartTriggerTag2 := text`
`StartTrigger Operation2`	Specify the operation that is applied on StartTriggerTag2, and StartTriggerTargetValue2 or StartTriggerTargetTag2. **Enter** **For** 0 Exact Equal (Tag EQU Target) 1 Trigger Level Equal (Tag = Target) 2 Not Equal (Tag != Target) 3 Less Than (Tag < Target) 4 Greater Than (Tag > Target) 5 Less Than or Equal To (Tag <= Target) 6 Greater Than or Equal To (Tag >= Target) 7 Positive Slope (slope of Tag is positive) 8 Negative Slope (slope of Tag is negative) 9 Bitwise OR ((Tag OR Target) = 0) 10 Bitwise OR ((Tag OR Target) != 0) 11 Bitwise AND ((Tag AND Target) = 0) 12 Bitwise AND ((Tag AND Target) != 0) 13 Bitwise XOR ((Tag XOR Target) = 0) 14 Bitwise XOR ((Tag XOR Target) != 0) Specify `StartTriggerOperation2 := number`
`StartTriggerTarget Type2`	Specify the type of the second start trigger target. Enter TargetValue or TargetTag. If you enter TargetValue, StartTriggerTargetValue2 is expected. Otherwise, StartTriggerTargetTag2 is expected. Specify `StartTriggerTargetType2 := text`
`StartTriggerTarget Value2`	Specify a target value if the StartTriggerTargetType2 is TargetValue. Enter a binary, octal, decimal, or hexadecimal integer number or enter a floating point number. Specify `StartTriggerTargetValue2 := text`
`StartTriggerTarget Tag2`	Specify a target tag if the StartTriggerTargetType is TargetTag. The tag must be one of the pen names. Specify `StartTriggerTargetTag2 := text`
`PreSampleType`	Define how pre-samples are specified. Enter Samples or TimePeriod. Specify `PreSampleType := text`
`PreSamples`	Specify the number of pre-samples (0...1000) if the PreSampleType is Samples. Specify a time period (0...(SamplePeriod * 1000)) that covers pre-samples if the PreSampleType is TimePeriod. Specify `PreSamples := number`
`StopTriggerType`	Specify the type of the stop trigger. Enter NoTrigger or Event Trigger. Specify `StopTriggerType := text`
`StopTriggerTag1`	Specify the tag name of the first trigger. The name must be one of the pen names. Specify `StopTriggerTag1 := text`

Attribute	Description
`StopTrigger Operation1`	Specify the operation that is applied on StopTriggerTag1 and StopTriggerTargetValue1 or StopTriggerTargetTag1. **Enter** **For** 0 Exact Equal (Tag EQU Target) 1 Trigger Level Equal (Tag = Target) 2 Not Equal (Tag != Target) 3 Less Than (Tag < Target) 4 Greater Than (Tag > Target) 5 Less Than or Equal To (Tag <= Target) 6 Greater Than or Equal To (Tag >= Target) 7 Positive Slope (slope of Tag is positive) 8 Negative Slope (slope of Tag is negative) 9 Bitwise OR ((Tag OR Target) = 0) 10 Bitwise OR ((Tag OR Target) != 0) 11 Bitwise AND ((Tag AND Target) = 0) 12 Bitwise AND ((Tag AND Target) != 0) 13 Bitwise XOR ((Tag XOR Target) = 0) 14 Bitwise XOR ((Tag XOR Target) != 0) Specify `StopTriggerOperation1 := number`
`StopTriggerTarget Type1`	Specify the type of the first stop trigger target. Enter TargetValue or TargetTag. If you specify TargetValue, StopTriggerTargetValue1 is expected. Otherwise, StopTriggerTargetTag1 is expected. Specify `StopTriggerTargetType1 := text`
`StopTriggerTarget Value1`	Specify a target value if the StopTriggerTargetType1 is TargetValue. Enter a binary, octal, decimal, or hexadecimal integer number or enter a floating point number. Specify `StopTriggerTargetValue1 := number`
`StopTriggerTarget Tag1`	Specify a target tag if the StopTriggerTargetType is TargetTag. The name must be one of the pen names. Specify `StopTriggerTargetTag1 := text`
`StopTriggerLogical Operation`	Specify a logical operation (AND or OR) that is performed on StopTriggerxxx1 and StopTriggerxxx2. StopTriggerxxx1 consists of StopTriggerTag1, StopTriggerOperation1, StopTriggerTargetType1, and StopTriggerTargetValue1 or StopTriggerTargetTag1. StopTriggerxxx2 consists of StopTriggerTag2, StopTriggerOperation2, StopTriggerTargetType2, and StopTriggerTargetValue2 or StopTriggerTargetTag2. Specify `StopTriggerLogicalOperation := text`
`StopTriggerTag2`	Specify the tag name of the second trigger. The name must be one of the pen names. Specify `StopTriggerTag2 := text`

Attribute	Description
`StopTrigger Operation2`	Specify the operation that is applied on StopTriggerTag2 and StopTriggerTargetValue2 or StopTriggerTargetTag2. **Enter** **For** 0 Exact Equal (Tag EQU Target) 1 Trigger Level Equal (Tag = Target) 2 Not Equal (Tag != Target) 3 Less Than (Tag < Target) 4 Greater Than (Tag > Target) 5 Less Than or Equal To (Tag <= Target) 6 Greater Than or Equal To (Tag >= Target) 7 Positive Slope (slope of Tag is positive) 8 Negative Slope (slope of Tag is negative) 9 Bitwise OR ((Tag OR Target) = 0) 10 Bitwise OR ((Tag OR Target) != 0) 11 Bitwise AND ((Tag AND Target) = 0) 12 Bitwise AND ((Tag AND Target) != 0) 13 Bitwise XOR ((Tag XOR Target) = 0) 14 Bitwise XOR ((Tag XOR Target) != 0) Specify `StopTriggerOperation2 := number`
`StopTriggerTarget Type2`	Specify the type of the second stop trigger target. Enter TargetValue or TargetTag. If you specify TargetValue, StopTriggerTargetValue2 is expected. Otherwise, StopTriggerTargetTag2 is expected. Specify `StopTriggerTargetType2 := text`
`StopTriggerTarget Value2`	Specify a target value if the StopTriggerTargetType2 is TargetValue. Enter a binary, octal, decimal, or hexadecimal integer number or enter a floating point number. Specify `StopTriggerTargetValue2 := number`
`StopTriggerTarget Tag2`	Specify a target tag if the StopTriggerTargetType is TargetTag. The name must be one of the pen names. Specify `StopTriggerTargetTag2 := text`
`PostSampleType`	Define how post-samples are specified. Enter Samples or TimePeriod. Specify `PostSampleType := text`
`PostSamples`	Specify the number of post-samples (0...1000) if the PostSampleType is Samples. Specify a time period (0...(SamplePeriod * 1000)) that covers post-samples if the PostSampleType is TimePeriod. Specify `PostSamples := number`
`TrendxVersion`	Specify the version of the Trend feature. Specify `TrendxVersion := number`

Specify a PEN Declaration

A TREND object can have as many as eight PEN declarations. A PEN declaration follows this structure.

```
PEN <pen_name> [(Attributes)];
END_PEN
```

Where:

Item	Identifies
`pen_name`	The pen.
`Attributes`	Attributes of the pen. Can also contain a description of the pen. Enclose in parenthesis. Separate each attribute with a comma (,).

Specify Attributes for a PEN Declaration

Specify these attributes for a PEN declaration.

Attribute	Description
`Description`	Provide information about the pen. Specify `Description := "text"`
`Color`	Specify the color of the line in RGB format. Enter the hex number for the color (16#0000_0000 – 16#00FF_FFFF). Specify `Color := hex_number`
`Visible`	Specify whether or not the line should be visible. Enter TRUE or FALSE. Specify `Visible := text`
`Width`	Specify the width of the line in pixels (1...10). Specify `Width := number`
`Type`	Specify the line type. Enter Analog, Digital, or Full-Width. Specify `Type := text`
`Style`	Specify the style of line. **Enter** **For** 0 ..... 1 … … … … 2 . . . . . 3 … . … . … . … 4 … .. … .. … .. Specify `Style := number`
`Marker`	Specify the line marker (0...83) Specify `Marker := number`

Attribute	Description
`Min`	Specify the minimum value for the pen. The minimum cannot be greater than or equal to the maximum. Specify `Min := number`
`Max`	Specify the maximum value for the pen. The maximum cannot be less than or equal to the minimum. Specify `Max := number`
`EngUnits`	Specify engineering units. For example, rpm, gallon, fps, and degrees. Specify `EngUnits := text`

TREND Guidelines

Keep these guidelines in mind when defining a trend.

- A trend can support as many as 8 PEN declarations.
- Export just the trend of a controller project by right-clicking on the trend in the Controller Organizer and selecting Export. This saves the trend as a .L5X file (XML format), which follows the same format as described above for the complete project .L5K file.
- To import a trend .L5X file into a controller project, right-click on the Trends in the Control Organizer and select Import.

TREND Example

```
TREND trend1 (SamplePeriod := 10,
              NumberOfCaptures := 1,
              CaptureSizeType := Samples,
              CaptureSize := 60000,
              StartTriggerType := No Trigger,
              StopTriggerType := No Trigger,
              TrendxVersion := 5.2)
Template :=
[208,207,17,224,161,177,26,225,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,62,0,3,0,254,255,9,0,6,0,0,0,0
,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,16,0,0,2,0,0,0,1,0,0,0,254,255,255,255,0,0,0,0,0,0,0
,0,255,255,255,255,255,255,....
PEN Local:1:I.CHA_Status (Color := 16#00ff_0000,
                                Visible := 1,
                                Width := 1,
                                Type := Analog,
                                Style := 0,
                                Marker := 0,
                                Min := 0.0,
                                Max := 100.0)
END_PEN
PEN Local:1:I.CHB_Status (Color := 16#0000_ff00,
                                Visible := 1,
                                Width := 1,
                                Type := Analog,
                                Style := 0,
                                Marker := 0,
                                Min := 0.0,
                                Max := 100.0)
END_PEN
END_TREND
```

Define Controller Objects

Introduction

This chapter explains how to enter project and configuration information in a complete import/export file.

Topic	Page
Define Controller Objects	179
Specify CONFIG Attributes	180
CONFIG Examples	184

Define Controller Objects

A CONFIG component defines controller objects and follows this structure.

```
CONFIG <object_name> [(Attributes)]
       [body]
END_CONFIG
```

Where:

Item	Identifies
`object_name`	The controller object.
`Attributes`	Attributes of the controller object. Can also contain a description of the controller object. Enclose in parenthesis. Separate each attribute with a comma (,).

Controller objects are optional. There can be only one of each controller object in an import/export file. Controller objects appear at the end of the import/export file.

Specify CONFIG Attributes

The attributes depend on the type on CONFIG object. Some objects do not have any attributes.

Object	Attribute	Description
ASCII	XONXOFFEnable	Specify whether to regulate the flow of incoming data. Enter 0 to disable XON/XOFF; enter 1 to enable XON/XOFF. Specify `XONXOFFEnable := value`
	DeleteMode	Specify the delete mode. Enter 0 for Ignore; enter 1 for CRT; or enter 2 for Printer. Specify `DeleteMode := value`
	EchoMode	Specify whether to echo data back to the device from which it was sent. Enter 0 to disable; enter 1 to enable. Specify `EchoMode := value`
	TerminationChars	Specify the characters that designate the end of a line. Specify `TerminationChars := value`
	AppendChars	Specify the characters to append to the end of a line. Specify `AppendChars := value`
	BufferSize	Specify the maximum size of the data array (1...65535 bytes) to send and receive. Specify `BufferSize := value`
ControllerDevice	none	none
CST	SystemTimeMasterID	Specify whether the controller is the coordinated system time master. Enter 16#0000 if the controller is not the CST master; enter 16#0001 if the controller is the CST master. Specify `CST := 16#value`

Object	Attribute	Description
DF1	DuplicateDetection	Specify whether to enable duplicate message detection, which ignores duplicate messages. Enter 0 to disable; enter 1 to enable. Specify `DuplicateDetection := value`
	ErrorDetection	Specify the error detection method. Enter BCC Error or CRC Error. Specify `ErrorDetection := text`
	EnbeddedResponseEnable	Specify the response method. Enter 0 to autodetect; enter 1 to enable. Specify `EnbeddedResponseEnable := value`
	DF1Mode	Specify the DF1 mode. Enter Pt to Pt, Master, or Slave. Specify `DF1Mode := value`
	ACKTimeout	Specify the time to wait for an acknowledgment to a message transmission. Enter an increment of 20ms (0...32767). Specify `ACKTimeout := value`
	NAKReceiveLimit	Specify the number of NAKS (0...127) the controller can receive in response to a message before stopping transmission. Specify `NAKReceiveLimit := value`
	ENQTransmit	Specify the number of inquiries (0...127) the controller sends after an ACK timeout. Specify `ENQTransmit := value`
	TransmitRetries	Specify the number of attempted retries (0...127) without getting an acknowledgement before the message is deemed undeliverable. Specify `TransmitRetries := value`
	StationAddress	Specify the current station link address (0...254). Specify `StationAddress := value`
	ReplyMessageWait	Specify the time the master waits after receiving an acknowledgment to a master-initiated message before polling the slave for a response. Enter an increment of 20ms (0...65535). Specify `ReplyMessageWait := value`
	PollingMode	Specify the polling mode. Enter on of these • 1 for Message Based (slave can initiate messages) • 2 for Message Based (slave cannot initiate messages) • 3 for Standard (multiple message transfer for node scan) • 4 for Standard (single message transfer per node scan) Specify `PollingMode := value`
	MasterMessageTransmit	Specify when the master transmits. Enter 0 to transmit between station polls; enter 1 to transmit in poll sequence. Specify `MasterMessageTransmit := value`
	NormalPollNodeFile	Specify the tag name of the structure that contains the normal poll node list. Or enter <NA>. The tag must specify Class = Standard. Specify `NormalPollNodeFile := value`
	NormalPollGroupSize	Specify the total number (0...255) of active stations polled from the poll list. Specify `NormalPollGroupSize := value`
	PriorityPollNodeFile	Specify the tag name of the structure that contains the priority poll node list. Or enter <NA>. The tag must specify Class = Standard. Specify `PriorityPollNodeFile := value`

Object	Attribute	Description
DF1 (continued)	`ActiveStationFile`	Specify the tag name of the structure that contains the status (active or non-active) of each node. Or enter <NA>. The tag must specify Class = Standard. Specify `ActiveStationFile := value`
	`SlavePollTimeout`	Specify the amount of time the master waits for an acknowledgement to a message sent to a slave. Enter an increment of 20ms (0...65535). Specify `SlavePollTimeout := value`
	`EOTSuppression`	Specify whether to enable EOT suppression. Enter 0 to disable; enter 1 to enable. Specify `EOTSuppression := value`
	`MaxStationAddress`	Specify the maximum station address (0...31). Specify `MaxStationAddress := value`
	`TokenHoldFactor`	Specify the token hold factor (1...4). Specify `TokenHoldFactor := value`
	`EnableStoreFwd`	For DF1 radio modem, specify whether to enable the store and forward feature. Enter 0 to disable; enter 1 to enable. Specify `EnableStoreFwd := value`
	`StoreFwdFile`	Specify the INT tag that holds the store and forward table. Specify `StoreFwdFile := text`
`ExtendedDevice`	none	none
`FaultLog`	none	none
`ICP`	none	none
`PCCC`	none	none
`Redundancy`	none	none
`SafetyController`	none	none

Object	Attribute	Description
SerialPort	BaudRate	Specify the communication rate for the serial port. Enter 110, 300 600, 1200, 2400, 4800, 9600, 19200, or 38400. Specify `BaudRate := value`
	Parity	Specify the parity setting for the serial port. Parity provides additional message-packet error detection. Enter None Parity, Even Parity, or Odd Parity. Specify `Parity := text`
	DataBits	Specify the number of bits per message packet. Enter 7 Data Bits or 8 Data Bits. Specify `DataBits := text`
	StopBits	Specify the number of stop bits to the device with which the controller is communicating. Enter 1 Stop Bit or 2 Stop Bit. Specify `StopBits := text`
	ComDriverId	Specify the type of serial driver. Enter DF1. Specify `ComDriverId := text`
	PendingComDriverId	Specify type of serial driver. Enter DF1. Specify `PendingComDriverId := text`
	RTSOffDelay	Specify a time delay to make sure the modem successfully transmits the entire message. Enter an increment of 20ms (0...32767). Normally leave at zero. Specify `RTSOffDelay := value`
	RTSSendDelay	Specify a time delay to let the modem prepare to transmit a message. Enter an increment of 20ms (0...32767). Specify `RTSSendDelay := value`
	ControlLine	Specify the mode in which the serial driver operates. Enter No Handshake, Full Duplex, Half Duplex without Continuous Carrier, or Half Duplex with Continuous Carrier. Specify `ControlLine := text`
	PendingControlLine	Specify the mode in which the serial driver operates. Enter No Handshake, Full Duplex, Half Duplex without Continuous Carrier, or Half Duplex with Continuous Carrier. Specify `PendingControlLine := text`
	RemoteModeChangeFlag	Specify whether there is a remote change. Enter 0 or 1. Specify `RemoteModeChangeFlag := value`
	PendingRemoteModeChange Flag	Specify whether there is a remote change. Enter 0 or 1. Specify `PendingRemoteModeChangeFlag := value`
	ModeChangeAttentionChar	Specify the mode change attention character. Specify `ModeChangeAttentionChar := value`
	PendingModeChange AttentionChar	Specify the mode change attention character. Specify `PendingModeChangeAttentionChar := value`

Object	Attribute	Description
SerialPort (continued)	SystemModeCharacter	Specify the system mode character. Specify `SystemModeCharacter := value`
	PendingSystemMode Character	Specify the system mode character. Specify `PendingSystemModeCharacter := value`
	UserModeCharacter	Specify the user mode character. Specify `SystemModeCharacter := value`
	PendingSystemMode Character	Specify the user mode character. Specify `PendingSystemModeCharacter := value`
	DCDWaitDelay	For DF1 radio modem, specify the delay in seconds (0...255). Specify this value if ControlLine is Half-Duplex and ContinuousCarrier is disabled. Specify `DCDWaitDelay := value`
UserMemory	none	none
WallClockTime	LocalTimeAdjustment	Specify any local time adjustment. Specify `LocalTimeAdjustment := value`
	TimeZone	Specify the time zone. Specify `TimeZone := value`

CONFIG Examples

This example shows a DF1 controller object.

```
CONFIG DF1
       DuplicateDetection := -1,
       ErrorDetection := BCC Error,
       EmbeddedResponseEnable := -1,
       DF1Mode := Pt to Pt,
       ACKTimeout := 50,
       NAKReceiveValue := 3,
       DF1ENQs := 3,
       DF1Retries := 3,
       StationAddress := 0,
       ReplyMessageWait := 50,
       PollingMode := 0,
       MasterMessageTransmit := 0,
       NormalPollNodeFile := NA,
       NormalPollGroupSize := 0,
       PriorityPollNodeFile := NA,
       ActiveStationFile := NA)
END_CONFIG
```

This example shows a SerialPort controller object.

```
CONFIG SerialPort
       (BaudRate := 19200,
       Parity := No Parity,
       DataBits := 8 Bits of Data,
       StopBits := 1 Stop Bit,
       ComDriverId := DF1,
       RTSOffDelay := 0,
       RTSSendDelay := 0,
       ControlLine := No Handshake,
       RemoteModeChangeFlag := 0,
       ModeChangeAttentionChar := 27,
       SystemModeCharacter := 83,
       UserModeCharacter := 85)
END_CONFIG
```

Notes:

Structure Tag and Comments in an Import/Export File

Introduction

This chapter explains how to structure the import/export file using commas (.CSV text file) or tabs (.TXT Unicode text file) to separate values in the file.

Topic	Page
Place Information in a .CSV or .TXT File	187
Specify a Tag Record	188
Specify a Comment Record	191
Specify an Alarm Message Record	193
Example .CSV File	194
Example .TXT File	196

Place Information in a .CSV or .TXT File

The structured import/export file contains these components of information.

Item	Identifies
Remark	Comment within the file.
TAG	Tag.
RCOMMENT	Rung comment.
TEXTBOX	Text box comment.

Internal File Comments

You can enter comments to document import files. The import process ignores these comments. You can place comments anywhere in an import/export file, except in names and descriptions. Enter comments by starting the line (record) with REMARK and a comma.

Specify a Tag Record

Each tag record defines a tag within a controller project. A TAG record includes this information.

Item	Identifies
Type	The type of tag. TAG tag ALIAS alias tag COMMENT tag operand component
Scope	What part of the project owns the tag. If no scope is specified, the scope is controller. If a scope is specified, it identifies the program or equipment phase.
Name	Name of the tag
Description	Description of the tag (optional)
Datatype	Datatype of the tag - use any valid datatype name
Specifier	Optional • For an alias, specifies base tag. • For a tag comment, specifies the tag name and member or bit.
Attributes	The attributes of the tag, as exported in the .L5K format. Define how the tag can be used and how it appears. Attributes do not include tag values.

TAG Type Record

Each TAG record defines a tag within a controller project.

TAG Structure with Commas

```
TAG,"Scope","Name","Description","Datatype","Specifier","Attributes"
```

TAG Structure with Tabs

```
TAG,"Scope"  "Name"  "Description"  "Datatype"  "Specifier"  "Attributes"
```

Specify tag dimensions on the Datatype.

To specify	Enter
1 dimension	[a]
2 dimensions	[a,b]
3 dimensions	[a,b,c]

The following example shows TAG records in a .CSV format.

7	TYPE	SCOPE	NAME	DESCRIPTION	DATATYPE	SPECIFIER	ATTRIBUTES		
8	TAG		Local:1:C		AB:1756_DI:C:0				
9	TAG		Local:1:I		AB:1756_DI:I:0				
10	TAG		Local:4:C		AB:1756_DO:C:0				
11	TAG		Local:4:I		AB:1756_DO:I:0				
12	TAG		Local:4:O		AB:1756_DO:O:0				
13	ALIAS		input_1			Local:1:I.Data.20	(RADIX := Decimal)		
14	TAG		MC_Reset		MC_Config				
15	ALIAS		output_light			Local:4:O.Data.12	(RADIX := Decimal)		
16	TAG		Reset4		MESSAGE		(MessageType := CIP Generic, RequestedLeng		
17	TAG		Write_MC_Cfg9		MESSAGE		(MessageType := CIP Generic, RequestedLeng		
18	TYPE	SCOPE	NAME	DESCRIPTION	DATATYPE	SPECIFIER	ATTRIBUTES		
19	TAG	MainProgram	HeartBeat		TIMER[2]				
20	TAG	MainProgram	HeartBeatWord		DINT		(RADIX := Decimal)		
21	TYPE	SCOPE	NAME	DESCRIPTION	DATATYPE	SPECIFIER	ATTRIBUTES		
22	TAG	Multicast_Configure	Configure_Multicast		BOOL		(RADIX := Decimal)		
23	TAG	Multicast_Configure	DoNothing		BOOL		(RADIX := Decimal)		
24	TAG	Multicast_Configure	MC_Cfg_TTL_over_1		BOOL		(RADIX := Decimal)		
25	TAG	Multicast_Configure	Read_Mcast		DINT		(RADIX := Decimal)		

ALIAS Type Record

Each ALIAS record defines an alias within a controller project.

ALIAS Structure with Commas

```
ALIAS,"Scope","Name","Description","Datatype","Specifier","Attributes"
```

ALIAS Structure with Tabs

```
ALIAS    "Scope"  "Name"  "Description"  "Datatype"  "Specifier"  "Attributes"
```

The following example shows ALIAS records in a .CSV format.

7	TYPE	SCOPE	NAME	DESCRIPTION	DATATYPE	SPECIFIER	ATTRIBUTES
8	ALIAS		input_1			Local:1:I.Data.20	(RADIX := Decimal)
9	ALIAS		output_light			Local:4:O.Data.12	(RADIX := Decimal)

COMMENT Type Record

Each COMMENT record defines a comment about a component of a tag, such as a bit member, structure member, or an array element.

COMMENT Structure with Commas

```
COMMENT,"Scope","Name","Description","Datatype","Specifier","Attributes"
```

COMMENT Structure with Tabs

```
COMMENT  "Scope"  "Name"  "Description","Datatype"  "Specifier"  "Attributes"
```

The following example shows COMMENT records in a .CSV format.

9	TYPE	SCOPE	NAME	DESCRIPTION	DATATYPE	SPECIFIER	ATTRIBUTES
10	TAG		array_1		DINT[10]		(RADIX := Decimal)
11	COMMENT		array_1	first element in array_1		array_1[0]	
12	TAG		timer_1		TIMER		
13	COMMENT		timer_1	timer_1 enable		timer_1.EN	

Specify a Comment Record

Each comment record defines a rung comment or text box in the controller project. This is different than the COMMENT type that defines a comment about a tag component. A comment record includes this information.

Item	Identifies
Type	The type of comment. RCOMMENT ladder rung comment TEXTBOX function block or sequential function chart comment
Scope	What part of the project owns the comment. A program or equipment phase must be specified.
Routine	Name of the routine.
Comment	Text of the comment.
Owning Element	For RCOMMENT entries, neutral text for the last instruction on the rung that owns the comment. If there is no element on the rung, the Owning Element is a semi-colon (;). By default, the Owning Element is used to match the comment to a rung on import. For a TEXTBOX entry of an attached text box, neutral text identifies the element the text box is attached to. The Owning Element contains the backing tag name and the full specifier of the element, including the absolute location of the element. Owning Element. For a TEXTBOX entry of a free-floating text box, this entry is blank.
Location	For RCOMMENT entries, the rung number of comment. The rung number in the Location column is used to match the comment to a rung if either the Owning Element is blank for that comment or if you override the import default by selecting Match all RLL rung comments by rung number only. For TEXTBOX entries, either the absolute location of free-floating text boxes or the relative location from the owning element of attached text boxes. For absolute locations, the location contains both the sheet number and the X and Y coordinates of the text box. For relative locations, the location contains only the X and Y coordinates.

An RCOMMENT record follows this format.

RCOMMENT Structure with Commas

```
RCOMMENT,"Scope","Routine","Comment","Owning Element","Location"
```

RCOMMENT Structure with Tabs

```
RCOMMENT  "Scope"  "Routine"  "Comment"  "Owning Element"  "Location"
```

A TEXTBOX record follows this format.

TEXTBOX Structure with Commas

```
TEXBOX,"Scope","Routine","Comment","Owning Element","Location"
```

TEXTBOX Structure with Tabs

```
TEXTBOX  "Scope"  "Routine"  "Comment"  "Owning Element"  "Location"
```

The following example shows comment records in a .CSV format.

118	TYPE	SCOPE	ROUTINE	COMMENT	OWNING_ELEMENT	LOCATION	
119	TEXTBOX	Motor_Starter_Progr	Motor_Starter_FBI	Gives simple conveyor control such as start, stop, jog, and $		60:20:Sheet1	
120	RCOMMENT	Motor_Starter_Progr	Motor_Starter_LD	If Job_PB = on, then jog the conve	OTE(Motor_Starter_LD.J	0	
121	TEXTBOX	Motor_Starter_Progr	Motor_Starter_SFC	SFC Execution Configuration$N$NLast Scan of Active Steps		360:40:00	
122	RCOMMENT	Motor_Starter_Progr	Nested_Motor_Sta	Conveyor Control$NThese rungs ar	NOP()	0	
123	TYPE	SCOPE	ROUTINE	COMMENT	OWNING_ELEMENT	LOCATION	
124	RCOMMENT	Sort_Program	Fill_Init_Numbers	Fills the Init_Numbers array with 1	OTU(Fill_Numbers)	0	
125	TYPE	SCOPE	ROUTINE	COMMENT	OWNING_ELEMENT	LOCATION	
126	RCOMMENT	Motor_Starter:AOI	Logic	If the Stop button is closed, the mo	OTE(RunCommand)	0	
127	RCOMMENT	Motor_Starter:AOI	Logic	The motor starts if the run commar	OTE(Out)	1	
128	RCOMMENT	Motor_Starter:AOI	Logic	If FaultTime is greater than 0, turn	OTE(CheckAuxContact)	2	
129	RCOMMENT	Motor_Starter:AOI	Logic	If CheckAuxContact is on, the rung	OTL(Fault)	3	
130	RCOMMENT	Motor_Starter:AOI	Logic	To clear the fault of the motor, turn	OTU(Fault)	4	

Specify an Alarm Message Record

An alarm tag can have several alarm message strings for different alarm conditions and in different languages. An alarm message record includes this information.

Item	Identifies
Type	The alarm message and its associated language as: ALMMSG:*language* Languages: EN-US (United States English), DE (Germany German), ES (Spain Spanish), FR (France French), IT (Italian), PT (Brazil Portuguese), JA (Japanese), KO (Korean), ZH (Chinese)
Scope	What part of the project owns the comment. A program or equipment phase must be specified.
Name	Name of the associated alarm tag.
Description	Text of the alarm message.
Datatype	The type of alarm. Specify ALARM_DIGITAL or ALARM_ANALOG.
Specifier	Specify the type of alarm. **Specify** **For** AM digital alar**m** HH high-high analog alarm H high analog alarm L low analog alarm LL low-low analog alarm POS rate-of-change positive analog alarm NEG rate-of change negative analog alarm

An ALMMSG record follows this format.

ALMMSG Structure with Commas

```
ALMMS:language,"Scope","Name","Description","Datatype","Specifier"
```

ALMMSG Structure with Tabs

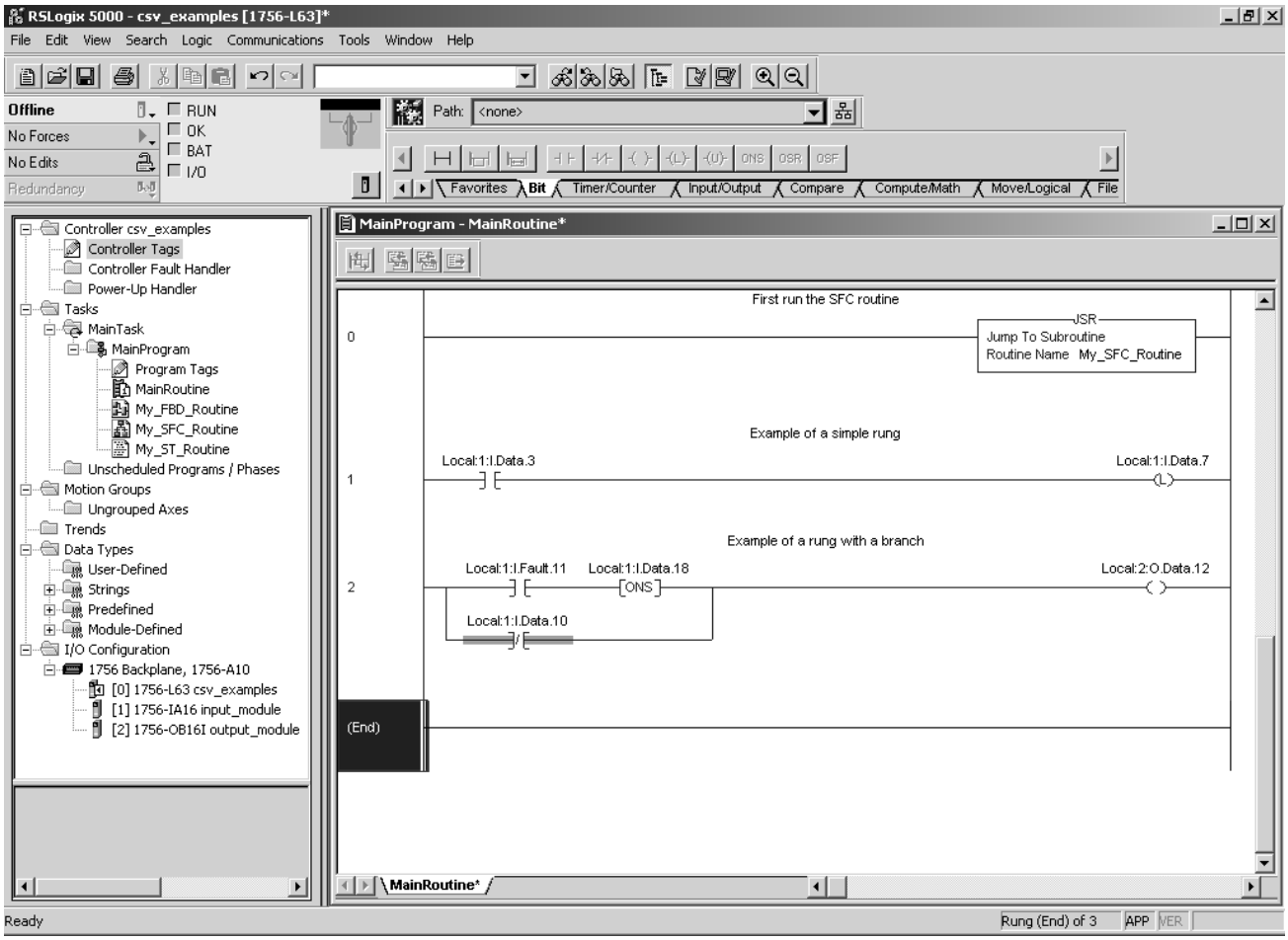
```
ALMMSG:language  "Scope"  "Name"  "Description"  "Datatype"  "Specifier"
```

The following example shows alarm message records in a .CSV format.

39	TAG	Motor_Starter_Program	ALMA_01		ALARM_ANALOG	
40	ALMMSG:en-us	Motor_Starter_Program	ALMA_01	high-message		HH
41	ALMMSG:fr	Motor_Starter_Program	ALMA_01	message in french		POS
42	ALMMSG:de	Motor_Starter_Program	ALMA_01	message in german		NEG
43	TAG	Motor_Starter_Program	ALMD_01		ALARM_DIGITAL	
44	ALMMSG:en-us	Motor_Starter_Program	ALMD_01	digital message		AM

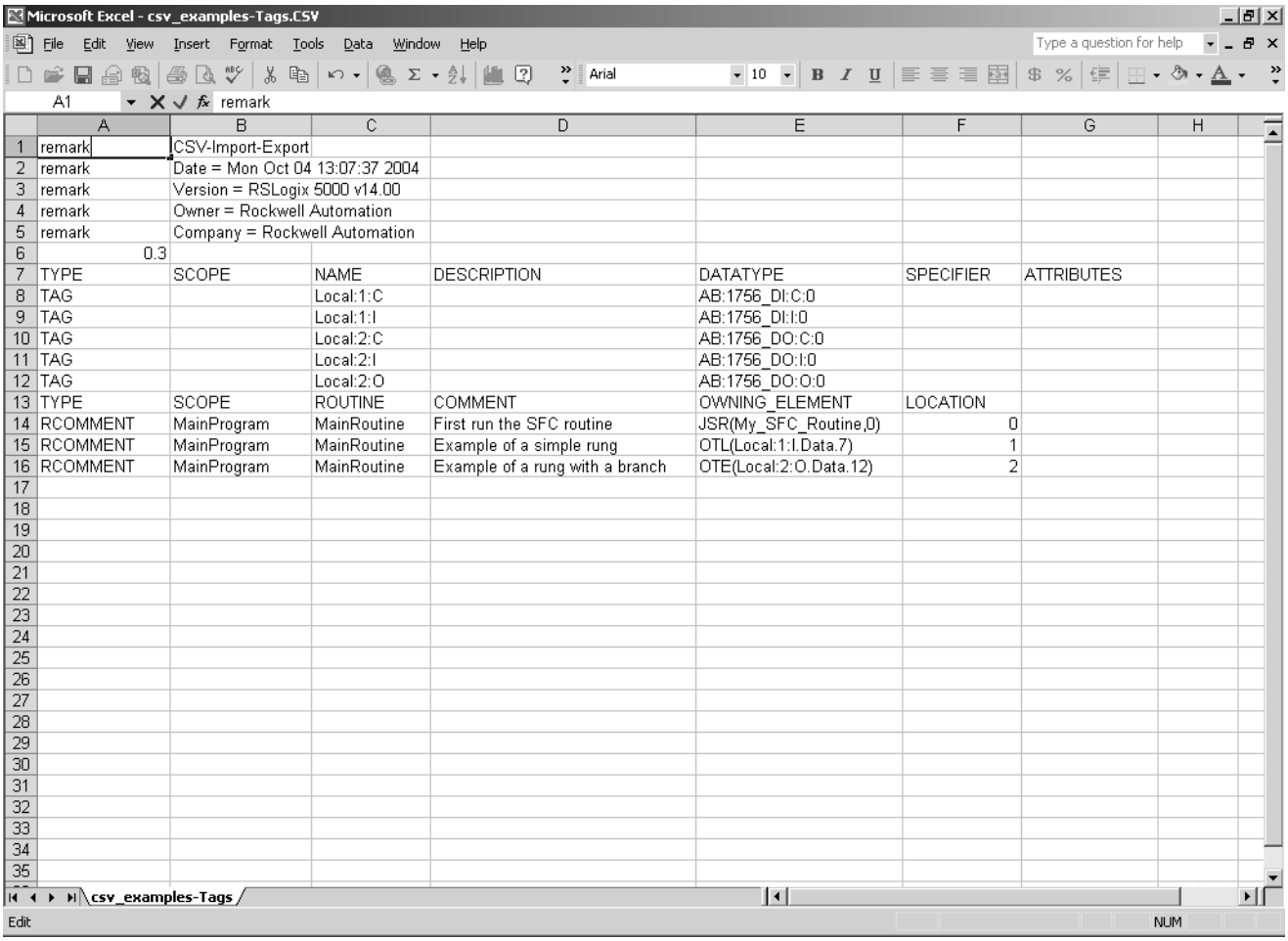
Example .CSV File

The following examples use this ladder file.



Export All Tags and Comments

An export of all tags and comments results in this .CSV file.



Microsoft Excel - csv_examples-Tags.CSV

File Edit View Insert Format Tools Data Window Help

A1 remark

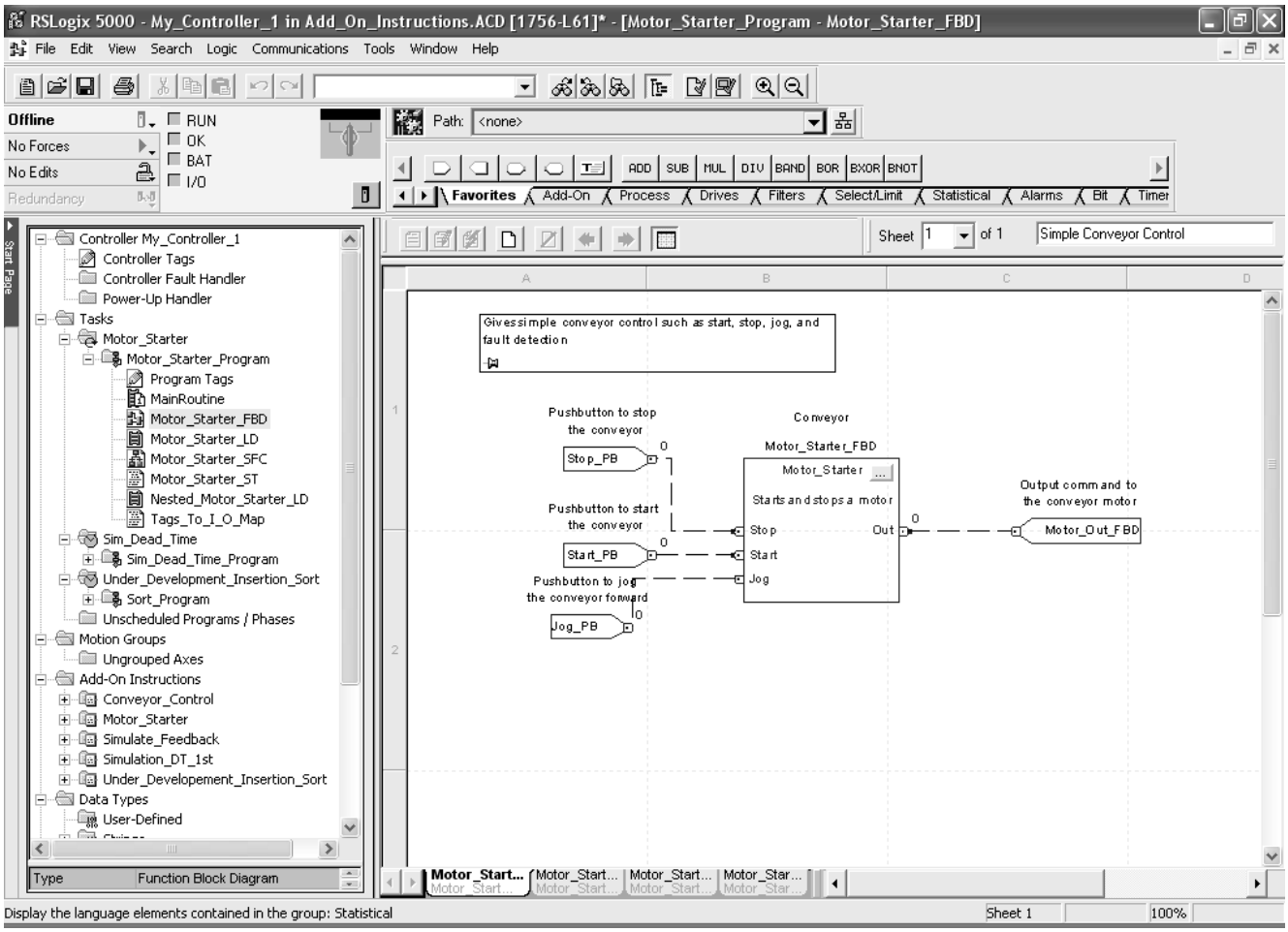
	A	B	C	D	E	F	G	H
1	remark	CSV-Import-Export						
2	remark	Date = Mon Oct 04 13:07:37 2004						
3	remark	Version = RSLogix 5000 v14.00						
4	remark	Owner = Rockwell Automation						
5	remark	Company = Rockwell Automation						
6	0.3							
7	TYPE	SCOPE	NAME	DESCRIPTION	DATATYPE	SPECIFIER	ATTRIBUTES	
8	TAG		Local:1:C		AB:1756_DI:C:0			
9	TAG		Local:1:I		AB:1756_DI:I:0			
10	TAG		Local:2:C		AB:1756_DO:C:0			
11	TAG		Local:2:I		AB:1756_DO:I:0			
12	TAG		Local:2:O		AB:1756_DO:O:0			
13	TYPE	SCOPE	ROUTINE	COMMENT	OWNING_ELEMENT	LOCATION		
14	RCOMMENT	MainProgram	MainRoutine	First run the SFC routine	JSR(My_SFC_Routine,0)	0		
15	RCOMMENT	MainProgram	MainRoutine	Example of a simple rung	OTL(Local:1:I.Data.7)	1		
16	RCOMMENT	MainProgram	MainRoutine	Example of a rung with a branch	OTE(Local:2:O.Data.12)	2		

csv_examples-Tags

Edit NUM

Example .TXT File

The following examples use the Motor_Starter_Program program file and exports the program tags.



Export Program Tags and Comments

An export of the Motor_Starter_Prorgam program tags and comments results in this .TXT file.

Add_On_Instructions-Motor_Starter_Program-Tags - Notepad

File Edit Format View Help

```
remark  "CSV-Import-Export"
remark  "Date = Mon Oct 16 13:45:21 2006"
remark  "Version = RSLogix 5000 v16.00"
remark  "Owner = Rockwell Automation"
remark  "Company = Rockwell Automation"
0.3
TYPE    SCOPE   NAME    DESCRIPTION     DATATYPE        SPECIFIER       ATTRIBUTES
TAG     Motor_Starter_Program   Action_000      ""      "SFC_ACTION"    ""      ""
TAG     Motor_Starter_Program   Conveyor_1_AuxContact    ""      "Simulate_Feedback"     ""      ""
TAG     Motor_Starter_Program   Conveyor_1_ClearFaultPB ""      "BOOL"  ""      "(RADIX := Decimal)"
TAG     Motor_Starter_Program   Conveyor_1_Entry_PE     ""      "BOOL"  ""      "(RADIX := Decimal)"
TAG     Motor_Starter_Program   Conveyor_1_JamClearPB   ""      "BOOL"  ""      "(RADIX := Decimal)"
TAG     Motor_Starter_Program   Conveyor_1_LD   "Starts and stops the conveyor" "Conveyor_Control"      ""      ""
TAG     Motor_Starter_Program   Conveyor_1_Out  ""      "BOOL"  ""      "(RADIX := Decimal)"
TAG     Motor_Starter_Program   Conveyor_1_Stop_PB      ""      "BOOL"  ""      "(RADIX := Decimal)"
TAG     Motor_Starter_Program   Jog_PB  "Pushbutton to jog the conveyor forward"        "BOOL"  ""      "(RADIX := Decimal)"
TAG     Motor_Starter_Program   Language        "LD = 0$NFBD = 1$NSFC = 2$NST = 3"      "DINT"  ""      "(RADIX := Decimal)"
TAG     Motor_Starter_Program   Motor_1_AuxContact      ""      "Simulate_Feedback"     ""      ""
TAG     Motor_Starter_Program   Motor_1_Clear_Fault_PB  ""      "BOOL"  ""      "(RADIX := Decimal)"
TAG     Motor_Starter_Program   Motor_Out_FBD   "Output command to the conveyor motor"  "BOOL"  ""      "(RADIX := Decimal)"
TAG     Motor_Starter_Program   Motor_Out_LD    "Output command to the conveyor motor"  "BOOL"  ""      "(RADIX := Decimal)"
TAG     Motor_Starter_Program   Motor_Out_SFC   "Output command to the conveyor motor"  "BOOL"  ""      "(RADIX := Decimal)"
TAG     Motor_Starter_Program   Motor_Out_ST    "Output command to the conveyor motor"  "BOOL"  ""      "(RADIX := Decimal)"
TAG     Motor_Starter_Program   Motor_Starter_FBD       "Conveyor"      "Motor_Starter" ""      ""
TAG     Motor_Starter_Program   Motor_Starter_LD        "Conveyor"      "Motor_Starter" ""      ""
TAG     Motor_Starter_Program   Motor_Starter_SFC       "Conveyor"      "Motor_Starter" ""      ""
TAG     Motor_Starter_Program   Motor_Starter_ST        "Conveyor"      "Motor_Starter" ""      ""
TAG     Motor_Starter_Program   Reset   ""      "BOOL"  ""      "(RADIX := Decimal)"
TAG     Motor_Starter_Program   Start_PB        "Pushbutton to start the conveyor"      "BOOL"  ""      "(RADIX := Decimal)"
TAG     Motor_Starter_Program   Step_000        ""      "SFC_STEP"      ""      ""
TAG     Motor_Starter_Program   Step_001        ""      "SFC_STEP"      ""      ""
TAG     Motor_Starter_Program   Step_002        ""      "SFC_STEP"      ""      ""
TAG     Motor_Starter_Program   Stop_PB "Pushbutton to stop the conveyor"       "BOOL"  ""      "(RADIX := Decimal)"
TAG     Motor_Starter_Program   Tran_000        ""      "BOOL"  ""      "(RADIX := Decimal)"
TAG     Motor_Starter_Program   Tran_001        ""      "BOOL"  ""      "(RADIX := Decimal)"
TAG     Motor_Starter_Program   Tran_002        ""      "BOOL"  ""      "(RADIX := Decimal)"
TAG     Motor_Starter_Program   UseConveyorSimulation   ""      "BOOL"  ""      "(RADIX := Decimal)"
TAG     Motor_Starter_Program   Use_Real_I_O    ""      "BOOL"  ""      "(RADIX := Decimal)"
TYPE    SCOPE   ROUTINE COMMENT OWNING_ELEMENT  LOCATION
TEXTBOX "Motor_Starter_Program" "Motor_Starter_FBD"     "Gives simple conveyor control such as start, stop, jog, and $R$Nfault detection"
""      60:20:Sheet1
RCOMMENT        "Motor_Starter_Program" "Motor_Starter_LD"      "If Job_PB = on, then jog the conveyor forward."        "OTE
(Motor_Starter_LD.Jog)" "0"
TEXTBOX "Motor_Starter_Program" "Motor_Starter_SFC"     "SFC Execution Configuration$N$NLast Scan of Active Steps = $R$NAutomatic Reset"
""      360:40
RCOMMENT        "Motor_Starter_Program" "Nested_Motor_Starter_LD"       "Conveyor Control$NThese rungs are an example of using the
Conveyor_Control Add-On Instruction."   "NOP()" "0"
```

Notes:

Structure the (.L5X) Partial Import/Export File Format

Introduction

This chapter explains the overall structure of the .L5X (Logix5000 XML) file that can store a portion of an RSLogix 5000 project. With RSLogix 5000 version 13, this includes ladder diagram logic fragments and the configuration for graphical trends.

Topic	Page
Place Information in a Ladder Rung .L5X File	202
Define a DataType Component	202
Define a Module Component	205
Define an Add-on Instruction Component	206
Define a Tag Component	209
Define a Program Component	210
Example Ladder Rung .L5X File	213
Place Information in a Trend .L5X File	214

The .L5X file is an ASCII file that is based on the format of the .L5K file but is structured using Extensible Markup Language (XML) tags. In addition to being able to open and modify the file .L5X file in a text editor, such as Notepad, you can also view the contents of the file in Microsoft Internet Explorer and other tools that work with XML files.

If you use	You see
A text editor, such as Notepad.	A text file, such as: <?xml version="1.0" encoding="UTF-8" standalone="yes"?> <RSLogix5000Content SchemaRevision="1.0" SoftwareRevision="13.00" TargetType="Rung" ContainsContext="true" Owner="Rockwell Automation, Rockwell Automation" ExportDate="Tue Mar 30 09:59:21 2004" ExportOptions="References Context ReferencesByUId RoutineLabels UIds AliasExtras IOTags NoStringData"> <Controller Use="Context" Name="test13" UId="f90ef335"> <DataTypes Use="Context" UId="d0c0ad05"> <DataType Name="SampleDT" Family="NoFamily" Class="User" UId="23e0ab2b"> <Members> <Member Name="Sample_DINT_Member" DataType="DINT" Dimension="0" Radix="Decimal" Hidden="false"> <Description> <![CDATA[This is a DINT member of the UDT]]> </Description> </Member> You can edit this file in the text editor.
An Internet browser, such as Internet Explorer.	An XML file, such as: <?xml version="1.0" encoding="UTF-8" standalone="yes" ?> – <RSLogix5000Content SchemaRevision="**1.0**" SoftwareRevision="**13.00**" TargetType="**Rung**" ContainsContext="**true**" Owner="**Rockwell Automation, Rockwell Automation**" ExportDate="**Tue Mar 30 09:59:21 2004**" ExportOptions="**References Context ReferencesByUId RoutineLabels UIds AliasExtras IOTags NoStringData**"> – <Controller Use="**Context**" Name="**test13**" UId="**f90ef335**"> + <DataTypes Use="**Context**" UId="**d0c0ad05**"> + <Tags Use="**Context**" UId="**d5a18dfb**"> + <Programs Use="**Context**" UId="**3238c36d**"> </Controller> </RSLogix5000Content> In the Internet browser, you can view only the file. Use the plus (+) and minus (-) signs to expand and collapse the viewable content. To edit the file, you must open the file in a text editor.

The examples in this chapter use Internet Explorer to display content.

You can create .L5X files for:

.L5X File	Description
Ladder rungs	To create the .L5X file for ladder rungs: 1. Select one or more rungs in a ladder routine. 2. Right-click on the selected rungs and select Export Rungs. The resulting .L5X files contains the rung logic, tag definitions, user-defined structures, and all associated descriptions. The files also contain any Add-On Instructions in the logic. To bring the contents of an .L5X file back into a project: 1. Navigate to where you want to import the rungs in a ladder routine. 2. Right-click and select Import Rungs. When you import an .L5X file, RSLogix 5000 software provides a list of the tags and user-defined structures in the .L5X file and lets rename them and their associated descriptions prior to the import process.
Trends Add-On Instructions User-defined data types	You can also select a trend, Add-On Instruction, or user-defined data type to export to an .L5X file: 1. Select the appropriate icon in the Controller Organizer of the project. 2. Right-click and select the export option. To import a trend, Add-On Instruction, or user-defined data type: 1. Select the appropriate icon in the Controller Organizer of the project. 2. Right-click and select the import option.

Identify Components in .L5X Files

Each component in an .L5X file has an associated UID (unique identifier). This identifier is a combination of letters and numbers and it links the associated component of the file with some object that is defined earlier. For example a tag definition uses a UID to link to a user-defined structure that is defined earlier in the project. A single UID can be defined only once in an .L5X file. You cannot reuse the same UID to define two components in the same file.

RSLogix 5000 software creates UIDs to provide an abstraction layer between definitions and their respective names. For example, the instructions refer to a UID for the tags that they use. By doing this, the tag can be renamed without having to search and replace all of the logic references within the rungs.

UIDs are not optional. They are required for each component in an .L5X file. See the rest of this chapter for descriptions of the supported component types.

Place Information in a Ladder Rung .L5X File

The .L5X file for ladder rungs uses this structure.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<RSLogix5000Content SchemaRevision="1.0" SoftwareRevision="13.00" TargetType="Rung"
  ContainsContext="true" Owner="Rockwell Automation, Rockwell Automation" ExportDate="Fri
  Jan 23 09:05:31 2004" ExportOptions="References Context ReferencesByUId RoutineLabels
  UIds AliasExtras IOTags NoStringData">
  <Controller Use="Context" Name="Test13" UId="f90ef335">
    <DataTypes Use="Context" UId="d0c0ad05">
    <Modules Use="Context" UId="cb15434a">
    <Tags Use="Context" UId="d5a18dfb">
    <Programs Use="Context" UId="3238c36d">
  </Controller>
</RSLogix5000Content>
```

Where:

Item	Identifies
Use	The use of the controller project. Specify Context or Target.
Name	The name of the controller project.
UID	The controller project with a unique combination of numbers and letters.
DataTypes	Data type definitions.
Modules	I/O module definitions.
AddOnInstructionDefintion	Add-On Instruction definitions.
Tags	Tag definitions.
Programs	Program and routine(s) containing the rung logic.

The first part of the .L5X file is the header that defines the version of the import/export feature. Following the header is the Controller component, which is the overall structure for an .L5X file.

Define a DataType Component

The DataType component defines the data types used in the section of rungs you export. Or you can select to export a single user-defined data type. The DataType component uses this structure.

```
<DataTypes Use="Context" UId="d0c0ad05">
  <DataType Name="SampleDT" Family="NoFamily" Class="User" UId="23e0ab2b">
    <Members>
  </DataType>
</DataTypes>
```

Specify a DataType

Each data type declaration within this component defines a data type and the members of that data type. Repeat this structure for each data type.

```
<DataType Name="SampleDT" Family="NoFamily" Class="User" UId="23e0ab2b">
  <Members>
    <Member Name="Sample_DINT_Member" DataType="DINT" Dimension="0"
      Radix="Decimal" Hidden="false">
      <Description>
        <![CDATA[ This is a DINT member of the UDT  ]]>
      </Description>
    </Member>
    <Member Name="ZZZZZZZZZZSampleDT1" DataType="SINT" Dimension="0"
      Radix="Decimal" Hidden="true" />
    <Member Name="Sample_BOOL_Member" DataType="BIT" Dimension="0"
      Radix="Decimal" Hidden="false" Target="ZZZZZZZZZZSampleDT1"
      BitNumber="0">
      <Description>
        <![CDATA[ This is a BOOL Member of the UDT  ]]>
      </Description>
    </Member>
  </Members>
</DataType>
```

Where:

Item	Identifies
Name	The name of the data type.
Family	Specify StringFamily for a string data type. Specify NoFamily for all other data types.
Class	Class of data type. Specify User for user-defined.
UID	The controller project with a unique combination of numbers and letters.

Specify a Member

Each member declaration within a data type declaration defines the members of that data type. Repeat this structure for each member.

```
<Members>
  <Member Name="Sample_DINT_Member" DataType="DINT" Dimension="0"
    Radix="Decimal" Hidden="false">
    <Description>
      <![CDATA[ This is a DINT member of the UDT ]]>
    </Description>
  </Member>
  <Member Name="ZZZZZZZZZZSampleDT1" DataType="SINT" Dimension="0"
    Radix="Decimal" Hidden="true" />
  <Member Name="Sample_BOOL_Member" DataType="BIT" Dimension="0"
    Radix="Decimal" Hidden="false" Target="ZZZZZZZZZZSampleDT1"
    BitNumber="0">
    <Description>
      <![CDATA[ This is a BOOL Member of the UDT ]]>
    </Description>
  </Member>
</Members>
```

Where:

Item	Identifies
Name	The name of the member.
DataType	The data type of the member, such as SINT, INT, DINT, REAL, BOOL.
Dimension	Specify 0 (not an array) or 1, 2, 3 if an array.
Radix	Specify decimal, hex, octal, binary, exponential, float, or ASCII.
Hidden	Whether the member is a hidden member of the structure.
UID	The controller project with a unique combination of numbers and letters.

DataType Example

```
<DataTypes Use="Context" UId="d0c0ad05">
  <DataType Name="SampleDT" Family="NoFamily" Class="User" UId="23e0ab2b">
    <Members>
      <Member Name="Sample_DINT_Member" DataType="DINT" Dimension="0"
        Radix="Decimal" Hidden="false">
        <Description>
          <![CDATA[ This is a DINT member of the UDT  ]]>
        </Description>
      </Member>
      <Member Name="ZZZZZZZZZZSampleDT1" DataType="SINT" Dimension="0"
        Radix="Decimal" Hidden="true" />
      <Member Name="Sample_BOOL_Member" DataType="BIT" Dimension="0"
        Radix="Decimal" Hidden="false" Target="ZZZZZZZZZZSampleDT1"
        BitNumber="0">
        <Description>
          <![CDATA[ This is a BOOL Member of the UDT  ]]>
        </Description>
      </Member>
    </Members>
  </DataType>
</DataTypes>
```

Define a Module Component

The Module component defines any modules associated with the section of rungs you export. For example, the Module component can contain I/O modules referenced by I/O tags, modules accessed by GSV/SSV instructions, or controllers referenced in consumed tags. The Module component uses this structure.

```
<Modules Use="Context" UId="cb15434a">
  <Module Use="Context" Name="Local" UId="42741be5" />
  <Module Use="Context" UId="29b0c934" />
  <Module Use="Context" UId="fcfdb37a" />
</Modules>
```

Where:

Item	Identifies
Use	The use of the module. Specify Context or Target.
Name	The name of the module.
UID	The controller project with a unique combination of numbers and letters.

Define an Add-on Instruction Component

Each AddOnInstructionDefinition component defines an Add-On Instruction. The AddOnInstructionDefinition component uses this structure.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
- <RSLogix5000Content SchemaRevision="1.0" SoftwareRevision="16.00" TargetType="AddOnInstructionDefinition"
    ContainsContext="true" Owner="Rockwell Automation, Rockwell Automation" ExportDate="Mon Oct 16 12:57:21 2006"
    ExportOptions="Context">
  - <Controller Use="Context" Name="My_Controller_1">
    - <AddOnInstructionDefinitions Use="Context">
      + <AddOnInstructionDefinition Use="Target" Name="Conveyor_Control" Revision="1.0" Vendor="Rockwell"
          ExecutePrescan="false" ExecutePostscan="false" ExecuteEnableInFalse="false" CreatedDate="2005-10-
          24T18:50:43.227Z" CreatedBy="NA\Mills" EditedDate="2005-12-13T20:13:00.825Z" EditedBy="NA\Mills"
          SoftwareRevision="v16.00">
      </AddOnInstructionDefinitions>
    </Controller>
  </RSLogix5000Content>
```

Where:

Item	Identifies
Use	The use of the Add-On Instruction. Specify Context or Target.
Name	The name of the Add-On Instruction.
Attributes	Attributes of the Add-On Instruction. Can also contain a description of the add-on instruction.
Parameters	Parameters of the Add-On Instruction.
LocalTags	Local tags of the Add-On Instruction.
Routine	Logic that comprises the Add-On Instruction. Logic can be relay ladder, function block, or structured text.

Add-on Instruction Example

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
- <RSLogix5000Content SchemaRevision="1.0" SoftwareRevision="16.00" TargetType="AddOnInstructionDefinition" ContainsContext="true"
    Owner="Rockwell Automation, Rockwell Automation" ExportDate="Mon Oct 16 12:57:21 2006" ExportOptions="Context">
  - <Controller Use="Context" Name="My_Controller_1">
    - <AddOnInstructionDefinitions Use="Context">
      - <AddOnInstructionDefinition Use="Target" Name="Conveyor_Control" Revision="1.0" Vendor="Rockwell" ExecutePrescan="false"
          ExecutePostscan="false" ExecuteEnableInFalse="false" CreatedDate="2005-10-24T18:50:43.227Z" CreatedBy="NA\Mills"
          EditedDate="2005-12-13T20:13:00.825Z" EditedBy="NA\Mills" SoftwareRevision="v16.00">
        - <Description>
            <![CDATA[ Starts and stops a conveyor  ]]>
          </Description>
        - <AdditionalHelpText>
          - <![CDATA[
              Use this instruction to start and stop a conveyor. Use the instruction together with a photoeye or other digital
              sensor that detects product at the entry to the conveyor.
              - If the Stop button is closed, the conveyor starts when product passes the entry sensor.
              - The conveyor stops when you press (open) the Stop button.
              - Use the Jog bit to jog the conveyor. The Jog bit overrides the Stop button.

              Automatically Turn Off the Conveyor
              You can configure the instruction to automatically turn off the conveyor if there's no product after a certain time.
              - In NoLoadTime, enter how long you want to wait for product. Enter the time in milliseconds.
              - The conveyor turns off if product doesn't show up within the NoLoadTime.
              -You must set NoLoadTime greater than 0 to automatically turn off the conveyor. Otherwise it keeps running until you press (open)

              Check for a Jam
              You can configure the instruction to automatically turn off the conveyor if product gets stuck at the entry.
              - In JamTime, enter how long to wait before signaling a jam. Enter the time in milliseconds.
              -The conveyor turns off and the Jam bit turns on if the input sensor stays on for JamTime.
              -You must set JamTime greater than 0 to signal a Jam. Otherwise the conveyor keeps running until you stop it.
              - To clear the Jam bit, turn on the JamClear bit.

              Watch for a Motor Fault
              You can also use the auxiliary contact of the conveyor's motor to make a fault happen if the motor doesn't start or stop.
              - In FaultTime, enter how long you want to wait for the contact to open or close. Enter the time in milliseconds.
              - The Fault bit turns on if the contact doesn't show that the motor started or stopped within the FaultTime.
              -You must set FaultTime greater than 0 to use the auxiliary contact. Otherwise the instruction doesn’t use the value of the auxili
              - To clear the Fault bit, turn on the ClearFault bit.

              In the LD and ST programming languages, this instruction doesn't show all its operands. Use other instructions to access the opera
              - In LD, use XIC and OTE instructions to read the value of the auxiliary contact tag and write it to the AuxContact bit.
              - In ST, use an assignment (:=) to set the AuxContact bit equal to the value of the auxiliary contact tag.
            ]]>
          </AdditionalHelpText>
        - <Parameters>
          - <Parameter Name="Stop" DataType="BOOL" Usage="Input" Radix="Decimal" Required="true" Visible="true">
            - <Description>
                <![CDATA[ Enter the tag for the stop pushbutton for the conveyor.  ]]>
              </Description>
              <DefaultData>0</DefaultData>
            </Parameter>
```

Example continues on next page.

Add-On Instruction example continues.

```
    + <Parameter Name="Start" DataType="BOOL" Usage="Input" Radix="Decimal" Required="true" Visible="true">
    + <Parameter Name="AuxContact" DataType="BOOL" Usage="Input" Radix="Decimal" Required="false" Visible="false">
    + <Parameter Name="Jog" DataType="BOOL" Usage="Input" Radix="Decimal" Required="false" Visible="false">
    + <Parameter Name="JamClear" DataType="BOOL" Usage="Input" Radix="Decimal" Required="false" Visible="false">
    + <Parameter Name="ClearFault" DataType="BOOL" Usage="Input" Radix="Decimal" Required="false" Visible="false">
    + <Parameter Name="Out" DataType="BOOL" Usage="Output" Radix="Decimal" Required="true" Visible="true">
    + <Parameter Name="Jam" DataType="BOOL" Usage="Output" Radix="Decimal" Required="false" Visible="true">
    + <Parameter Name="Fault" DataType="BOOL" Usage="Output" Radix="Decimal" Required="false" Visible="true">
    + <Parameter Name="JamTime" DataType="DINT" Usage="Input" Radix="Decimal" Required="false" Visible="false">
    + <Parameter Name="FaultTime" DataType="DINT" Usage="Input" Radix="Decimal" Required="false" Visible="false">
    + <Parameter Name="NoLoadTime" DataType="DINT" Usage="Input" Radix="Decimal" Required="false" Visible="false">
    </Parameters>
  - <LocalTags>
    + <LocalTag Name="NoLoadTimer" DataType="TIMER">
    + <LocalTag Name="JamTimer" DataType="TIMER">
    + <LocalTag Name="Motor_Starter" DataType="Motor_Starter">
    + <LocalTag Name="OK_To_Run" DataType="BOOL" Radix="Decimal">
    + <LocalTag Name="CheckAuxContact" DataType="BOOL" Radix="Decimal">
    + <LocalTag Name="CheckJam" DataType="BOOL" Radix="Decimal">
    </LocalTags>
  - <Routines>
    - <Routine Name="Logic" Type="RLL">
      - <Description>
          <![CDATA[ Runs a conveyor based on start and stop inputs  ]]>
        </Description>
      - <RLLContent>
        - <Rung Number="0" Type="N">
          - <Text>
              <![CDATA[ XIC(Stop)XIO(Jam)XIO(NoLoadTimer.DN)OTE(OK_To_Run);  ]]>
            </Text>
          </Rung>
        - <Rung Number="1" Type="N">
          - <Text>
              <![CDATA[ XIC(Jog)OTE(Motor_Starter.Jog);  ]]>
            </Text>
          </Rung>
        + <Rung Number="2" Type="N">
        + <Rung Number="3" Type="N">
        + <Rung Number="4" Type="N">
        + <Rung Number="5" Type="N">
        + <Rung Number="6" Type="N">
        + <Rung Number="7" Type="N">
        + <Rung Number="8" Type="N">
        + <Rung Number="9" Type="N">
        </RLLContent>
      </Routine>
    </Routines>
  - <Dependencies>
      <Dependency Type="AddOnInstructionDefinition" Name="Motor_Starter" />
    </Dependencies>
  </AddOnInstructionDefinition>
</AddOnInstructionDefinitions>
```

Define a Tag Component

The Tag component defines the tags, either associated with the section of rungs you selected or within the program you selected. The Tag component uses this structure.

```
<Tags Use="Context" UId="d5a18dfb">
  <Tag Name="Sample_Tag" UId="3ad464e2" TagType="Base" DataType="SampleDT">
</Tags>
```

Where:

Item	Identifies
Use	The use of the tags. Specify Context or Target.
Tag Name	The name of the tag.
UID	The tag with a unique combination of numbers and letters.
TagType	Specify Alias or Base.
DataType	The data type of the tag, such as SINT, INT, DINT, REAL, BOOL.
Radix	Specify decimal, hex, octal, binary, exponential, float, or ASCII.
AliasFor	The base tag for an alias tag.

Within the .L5X file, Tag declarations before the Program component are for controller-scope tags. Tag declarations within a Program component are program-scope tags for that program.

Tag Example

```
<Tags Use="Context" Id="d5a18dfb">
  <Tag Name="Estop_Disabled" Id="a07e31ed" TagType="Alias" DataType="BOOL" Radix="Decimal"
    AliasFor="Local:1:I.Data.0">
    <Description>
      <![CDATA[ No Estop pressed ]]>
    </Description>
    <Data>00</Data>
  </Tag>
  <Tag Name="CN1_M" Id="10ff31dc" TagType="Alias" DataType="BOOL" Radix="Decimal"
    AliasFor="Local:2:O.Data.0">
    <Description>
      <![CDATA[ Conveyor CN1 Motor Starter Output ]]>
    </Description>
    <Data>00</Data>
  </Tag>
  <Tag Name="Local:2:O" Id="df5ca2c0" TagType="Base" DataType="AB:1756_DO:O:0" IO="true">
    <Data>00 00 00 00</Data>
    <ForceData>00 00 00 00 00 00 00 00 00 00 00 00</ForceData>
  </Tag>
  <Tag Name="Local:1:I" Id="547c3ce8" TagType="Base" DataType="AB:1756_DI:I:0" IO="true">
    <Data>00 00 00 00 00 00 00 00</Data>
    <ForceData>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</ForceData>
  </Tag>
</Tags>
```

Define a Program Component

The Program component defines the programs used in the section of rungs you export. The Program component uses this structure.

```
<Programs Use="Context" Id="3238c36d">
  <Program Use="Context" Name="MainProgram" Id="d06d4be1">
</Programs>
```

Specify a Program

Each program declaration within this component defines a program and the routines within that program. Repeat this structure for each program.

```
<Program Use="Context" Name="MainProgram" Id="d06d4be1">
  <Tags Use="Context" Id="f54231b5">
  <Routines Use="Context" Id="4fca5a03">
</Program>
```

Where:

Item	Identifies
Use	The use of the program. Specify Context or Target.
Name	The name of the program.
UID	The program with a unique combination of numbers and letters.

Specify a Routine

Each routine declaration within a program declaration defines the routines of that program. Repeat this structure for each routine.

```
<Routine Use="Context" Name="MainRoutine" UId="80080377">
  <RLLContent Use="Context">
    <Rung Use="Target" Number="0" Type="N" UId="cc69634c">
      <Text>
        <![CDATA[ XIC(@3ad464e2@.Sample_BOOL_Member)OTE
        (@3ad464e2@.Sample_DINT_Member.0);  ]]>
      </Text>
    </Rung>
  </RLLContent>
</Routine>
```

Where:

Item	Identifies
Use	The use of the routine or rung. Specify Context or Target.
Name	The name of the routine.
Number	The rung number.
Type	The type of rung, such as N for normal or I for insert.
UID	The routine or rung with a unique combination of numbers and letters.

Program Example

```
<Programs Use="Context" UId="3238c36d">
  <Program Use="Context" Name="MainProgram" UId="d06d4be1">
    <Tags Use="Context" UId="f54231b5">
      <Tag Name="CN1" UId="2e15ff86" TagType="Base" DataType="Conveyor_Type">
        <Description>
          <![CDATA[ Conveyor CN1  ]]>
        </Description>
        <Data>00 00 00 00 00 00 00 00 A0 86 01 00 00 00 00 00 00 00 00 00 D0 07 00 00 00 00 00 00 00 00
          00 00 88 13 00 00 00 00 00 00 00 00 00 00 88 13 00 00 00 00 00 00</Data>
      </Tag>
    </Tags>
    <Routines Use="Context" UId="4fca5a03">
      <Routine Use="Context" Name="MainRoutine" UId="80080377">
        <RLLContent Use="Context">
          <Rung Use="Target" Number="8" Type="N" UId="24cc887b">
            <Text>
              <![CDATA[ XIO(@a07e31ed@)XIC(@2e15ff86@.Motor_Run)XIO(@2e15ff86@.Motor_Fault)XIO
              (@2e15ff86@.Jam_Fault_Entry_PE)XIO(@2e15ff86@.Jam_Fault_Exit_PE)OTE(@10ff31dc@);  ]]>
            </Text>
          </Rung>
        </RLLContent>
      </Routine>
    </Routines>
  </Program>
</Programs>
```

Example Ladder Rung .L5X File

This example shows all the ladder rung components in one .L5X file.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<RSLogix5000Content SchemaRevision="1.0" SoftwareRevision="13.00" TargetType="Rung" ContainsContext="true"
  Owner="Rockwell Automation, Rockwell Automation" ExportDate="Fri Jan 23 09:05:31 2004" ExportOptions="References
  Context ReferencesByUId RoutineLabels UIds AliasExtras IOTags NoStringData">
  <Controller Use="Context" Name="Test13" UId="f90ef335">
    <DataTypes Use="Context" UId="d0c0ad05">
      <DataType Name="Conveyor_Type" Family="NoFamily" Class="User" UId="23e0ab2b">
        <Description>
          <![CDATA[ Parameters associated with conveyor operation ]]>
        </Description>
        <Members>
          <Member Name="ZZZZZZZZZZConveyor_T0" DataType="SINT" Dimension="0" Radix="Decimal"
            Hidden="true" />
          <Member Name="Motor_Run" DataType="BIT" Dimension="0" Radix="Decimal" Hidden="false"
            Target="ZZZZZZZZZZConveyor_T0" BitNumber="0">
            <Description>
              <![CDATA[ Run The Motor ]]>
            </Description>
          </Member>
        </Members>
      </DataType>
    </DataTypes>
    <Modules Use="Context" UId="cb15434a">
      <Module Use="Context" Name="Local" UId="42741be5" />
      <Module Use="Context" UId="29b0c934" />
      <Module Use="Context" UId="fcfdb37a" />
    </Modules>
    <Tags Use="Context" UId="d5a18dfb">
      <Tag Name="CN1_M" UId="10ff31dc" TagType="Alias" DataType="BOOL" Radix="Decimal"
        AliasFor="Local:2:O.Data.0">
        <Description>
          <![CDATA[ Conveyor CN1 Motor Starter Output ]]>
        </Description>
        <Data>00</Data>
      </Tag>
    </Tags>
    <Programs Use="Context" UId="3238c36d">
      <Program Use="Context" Name="MainProgram" UId="d06d4be1">
        <Tags Use="Context" UId="f54231b5">
          <Tag Name="CN1" UId="2e15ff86" TagType="Base" DataType="Conveyor_Type">
            <Description>
              <![CDATA[ Conveyor CN1 ]]>
            </Description>
            <Data>00 00 00 00 00 00 00 00 A0 86 01 00 00 00 00 00 00 00 00 00 D0 07 00 00 00 00 00 00 00 00
              00 00 88 13 00 00 00 00 00 00 00 00 00 00 88 13 00 00 00 00 00 00</Data>
          </Tag>
        </Tags>
```

```
<Programs Use="Context" UId="3238c36d">
  <Program Use="Context" Name="MainProgram" UId="d06d4be1">
    <Tags Use="Context" UId="f54231b5">
      <Tag Name="CN1" UId="2e15ff86" TagType="Base" DataType="Conveyor_Type">
        <Description>
          <![CDATA[ Conveyor CN1  ]]>
        </Description>
        <Data>00 00 00 00 00 00 00 00 A0 86 01 00 00 00 00 00 00 00 00 00 D0 07 00 00 00 00 00 00 00 00
          00 00 88 13 00 00 00 00 00 00 00 00 00 00 88 13 00 00 00 00 00 00</Data>
      </Tag>
    </Tags>
    <Routines Use="Context" UId="4fca5a03">
      <Routine Use="Context" Name="MainRoutine" UId="80080377">
        <RLLContent Use="Context">
          <Rung Use="Target" Number="8" Type="N" UId="24cc887b">
            <Text>
              <![CDATA[ XIO(@a07e31ed@)XIC(@2e15ff86@.Motor_Run)XIO(@2e15ff86@.Motor_Fault)XIO
              (@2e15ff86@.Jam_Fault_Entry_PE)XIO(@2e15ff86@.Jam_Fault_Exit_PE)OTE(@10ff31dc@);  ]]
            </Text>
          </Rung>
        </RLLContent>
      </Routine>
    </Routines>
  </Program>
</Programs>
</Controller>
</RSLogix5000Content>
```

Place Information in a Trend .L5X File

The .L5X file for trends contains these components.

Component	Identifies
Controller	Name of the controller.
Trend	The selected trends.

The trend file uses this structure.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<RSLogix5000Content SchemaRevision="1.0" SoftwareRevision="13.00" TargetType="Tren
  ContainsContext="true" Owner="Rockwell Automation, Rockwell Automation" ExportD
  Jul 16 14:36:33 2003" ExportOptions="Context">
  <Controller Use="Context" Name="Conveyor">
    <Trends Use="Context">
      <Trend Use="Target" Name="Conveyor_Operation" SamplePeriod="10"
        NumberOfCaptures="1" CaptureSizeType="Samples" CaptureSize="60000"
        StartTriggerType="No Trigger" StopTriggerType="No Trigger" TrendxVersion="5.2
    </Trends>
  </Controller>
</RSLogix5000Content>
```

Specify a Trend

Each trend declaration within this component defines a trend and the pens within that trend. Repeat this structure for each trend.

```
<Trends Use="Context">
  <Trend Use="Target" Name="Conveyor_Operation" SamplePeriod="10"
    NumberOfCaptures="1" CaptureSizeType="Samples" CaptureSize="60000"
    StartTriggerType="No Trigger" StopTriggerType="No Trigger" TrendxVersion="5.2">
    <Pens>
  </Trend>
</Trends>
```

Where:

Item	Identifies
Use	The use of the trend. Specify Context or Target.
Name	The name of the trend.

Trend Example

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<RSLogix5000Content SchemaRevision="1.0" SoftwareRevision="13.00" TargetType="Trend"
  ContainsContext="true" Owner="Rockwell Automation, Rockwell Automation" ExportDate="Wed
  Jul 16 14:36:33 2003" ExportOptions="Context">
  <Controller Use="Context" Name="Conveyor">
    <Trends Use="Context">
      <Trend Use="Target" Name="Conveyor_Operation" SamplePeriod="10"
        NumberOfCaptures="1" CaptureSizeType="Samples" CaptureSize="60000"
        StartTriggerType="No Trigger" StopTriggerType="No Trigger" TrendxVersion="5.2">
        <Pens>
          <Pen Name="M_T1" Color="16#00ff_0000" Visible="true" Width="1"
            Type="Analog" Style="0" Marker="0" Min="0.0" Max="100.0" />
          <Pen Name="MA_T1" Color="16#0000_ff00" Visible="true" Width="1"
            Type="Analog" Style="0" Marker="0" Min="0.0" Max="100.0" />
          <Pen Name="PE_T1_A" Color="16#0000_00ff" Visible="true" Width="1"
            Type="Analog" Style="0" Marker="0" Min="0.0" Max="100.0" />
          <Pen Name="FLT_T1" Color="16#00ff_00ff" Visible="true" Width="1"
            Type="Analog" Style="0" Marker="0" Min="0.0" Max="100.0" />
        </Pens>
      </Trend>
    </Trends>
  </Controller>
</RSLogix5000Content>
```

Notes:

Considerations for Using Microsoft Excel to Edit a .CSV File

Introduction

This appendix describes how using Microsoft Excel can affect a .CSV file.

IMPORTANT To edit the .CSV file, it is recommended that you use a database program tool, such as Microsoft Access, or a raw text editor. Many other desktop tools, such as Microsoft Word or Excel, might change the structure of the .CSV file and cause an import of the file to fail.

Recommendations

You can use Microsoft Excel to edit your .CSV tag file.

- Use single quotes instead of double quotes within descriptions and comments.
- Do not create descriptions or comments that consist only of numbers, have leading zeros, or have a leading symbol that Microsoft Excel treats specially. For example, do not create descriptions like:

  002
  +2
  =2
  -2
  .0

- Do not create descriptions or comments that start with a +, -, or = symbol. Even if you add text after the symbol, Excel displays `#NAME?` in the cell.

RSLogix 5000 Data Transformations

When RSLogix 5000 programming software exports tags, it performs these conversions.

Original Content	Content in .CSV File After Export
'	$'
"	$Q
newline	NL
tab	$T
$	$$

Microsoft Excel Data Transformation

When you open the exported .CSV file in Excel, these conversions occur.

Original Content	Content in .CSV File After Export	Content After Opening in Excel	Content After Saving from Excel	Details
.0	".0"	0	0	RSLogix 5000 addresses this as the specifier for a tag. If you enter this as an entire comment, you lose any preceding period (.). If you enter any text before or after this, Excel maintains the content.
=2	"=2"	2	2	If you enter this as an entire comment, you lose any preceding equal sign (=). If you enter any text before or after this, Excel maintains the content.
+2	"+2"	2	2	If you enter this as an entire comment, you lose any preceding plus sign (+). If you enter any text before or after this, Excel maintains the content.
002	"002"	2	2	If you enter this as an entire comment, you lose any preceding zeros. If you enter any text before or after this, Excel maintains the content.
test string	"test string"	test string	test string	Excel puts quotes around cell contents only if there is an embedded comma. RSLogix 5000 always places double quotes around text. But RSLogix 5000 can still handle the description without quotes.
"test string"	"$"test string$""	$test string$""	"$test string$"""""	Both Excel and RSLogix 5000 alter content when it includes a dollar sign ($).
has "quoted text" within string	"has $"quoted text$" within string"	has $quoted text$" within string"	"has $quoted text $"" within string"""	Both Excel and RSLogix 5000 alter content when it includes a dollar sign ($).
this has 'embedded' text	this has $'embedded$' text	this has $'embedded$' text	this has $'embedded$' text	Single quotes work fine in both software packages.

Original Content	Content in .CSV File After Export	Content After Opening in Excel	Content After Saving from Excel	Details
+text	"+text"	#NAME?	#NAME?	Do not start a description or comment with a plus sign (+).
-text	"-text"	#NAME?	#NAME?	Do not start a description or comment with a minus sign (-).
=text	"=text"	#NAME?	#NAME?	Do not start a description or comment with an equal sign (=).

Notes:

Import/Export Revision History

Introduction

This appendix contains a history of enhancements made to the import/export feature **since** version 1.1 (major revision 1, minor revision 1) that was included with RSLogix 5000 programming software, version 8.0.

These releases of the import/export feature correspond to these releases of RSLogix 5000 software.

RSLogix 5000 Version	Import/Export Version
16.xx	2.7
15.xx	2.6
13.xx	2.4
12.xx	2.3
11.xx	2.2
10.xx	2.1
9.00	2.0
5.02	1.2
8.xx, 7.xx, 6.xx, 2.xx	1.1
1.23, 1.21	1.0
1.11, 1.10	0.4

Topic	Page
Backward compatibility	222
Import/export version 2.6 (RSLogix 5000 software version 15)	223
Import/export version 2.4 (RSLogix 5000 software version 13)	223
Import/export version 2.3 (RSLogix 5000 software version 12)	224
Import/export version 2.2 (RSLogix 5000 software version 11	224
Import/export version 2.1 (RSLogix 5000 software version 10)	225

Topic	Page
Import/export version 2.0 (RSLogix 5000 software version 9)	227
Motion changes to support the SERCOS protocol	228
Import/export version 1.1 (RSLogix programming software version 8)	232

Backward Compatibility

The import/export feature supports backward compatibility for import operations. This means that the RSLogix 5000 programming software can import .L5K files that were generated by a previous version of the programming software. In some cases, an older .L5K file might not correctly import into newer version of the programming software. The revision history in this appendix will list any conditions when backward compatibility for an import operation does not work as expected.

The import/export feature does not support backward compatibility for export operations. This means that older version of the RSLogix 5000 programming software cannot read .L5K files that were created with newer versions of the programming software.

Each version of the RSLogix 5000 programming software exports .L5K files with a specific import/export version number. The RSLogix 5000 programming software imports any .L5K file with the same major revision number and the same or lower minor revision number. The major revision number increments when there are conditions such that the programming software cannot support backward compatibility for import operations. The minor revision number increments whenever there is a change in the file (a new module, an attribute is added, the set of options for an attribute is changed) that does not affect backward compatibility for import operations.

IMPORTANT Be careful when copying and pasting between versions of .L5K files. Do not paste objects from an older .L5K file into a newer version.

Import/Export Version 2.6 RSLogix 5000 Version 15

Version 2.6 (major revision 2, minor revision 6) of the import/export feature that is included with RSLogix 5000 programming software, version 15 included these major enhancements.

- Support for the 1769-L32C, 1769-L32CR CompactLogix and 1768-L43 CompactLogix controllers.

  This release also removed support for the 1756-L1 ControlLogix, 1794-L33 FlexLogix, 1769-L20 CompactLogix, 1769-L30 CompactLogix, and PowerFlex 700 S controllers.

- Equipment Phase program type and its relay ladder and structured text instructions.
- ControlLogix and SoftLogix controllers now support 100 programs per task.
- Information about when an imported file modifies a project such that you cannot go online and access a previously downloaded controller.
- Additional values for the Mode attribute of a MODULE component.
- New SERCOS IDN Read and SERCOS IDN Write message types.
- New motion AXIS_GENERIC_DRIVE type.
- Removal of the DescriptionWidth parameter from the STEP, TRANSITION, and STOP components in SFC logic.
- Addition of an Attributes column to the .CSV format for exported tags.

Import/Export Version 2.4 RSLogix 5000 Version 13

Version 2.4 (major revision 2, minor revision 4) of the import/export feature that is included with RSLogix 5000 programming software, version 13 included these major enhancements.

- Support for new controllers.
- ExtendedProp section to MODULE data.
- Support for new TAG attributes.

  Attributes can be in any order in an import/export file. The order shown in this document is the order the attributes export.

- Support for a TREND object in the import/export .L5K file.
- New MCSV instruction in ladder logic (chapter 4) and structured text.
- Online editing support for structured text and sequential function chart logic.
- Updated CSV format now includes rung comments.
- New .L5X format for partial import/export of ladder rungs, tags, and trends.

Import/Export Version 2.3 RSLogix 5000 Version 12

Version 2.3 (major revision 2, minor revision 3) of the import/export feature that is included with RSLogix 5000 programming software, version 12.01 included these major enhancements.

- The structured text component changed from STX_ROUTINE to ST_ROUTINE. The LanguageType attribute in SFC routines for embedded structured text also changed from STX to ST.
- Support for new controllers.
- Addition of the ControlNetSignature attribute to the MODULE component.
- Addition of the ProgrammaticallySendEventTrigger attribute to the TAG component.
- New COORDINATE_SYSTEM tag.
- Addition of several new attributes to the axis tag types.
- Addition of DisableFlag attribute to the PROGRAM component.
- Addition of EventTrigger and EventTag attributes to the TASK component to support Event tasks.
- New EVENT, IOT, MCCD, MCCM, MCLM, MCS, MCSD, and MCSR instructions in ladder logic and structured text.
- Addition of information regarding the LOGIC block when exporting online function block logic.
- Addition of new modules and their valid CommMethod and ConfigMethod values.

Import/Export Version 2.2 RSLogix 5000 Version 11

Version 2.2 (major revision 2, minor revision 2) of the import/export feature that is included with RSLogix 5000 programming software, version 11.10 included these major enhancements.

- Support for the 1756-L63 controller.
- New controller attributes to support sequential function charts.
- Corrected the DATATYPE attributes and added the FamilyType attribute.
- Additional information for the CompatibleModule and KeyMask attributes of the MODULE component.
- Addition of RSNetWorxFileName attribute to the MODULE component.
- Addition of SFC_ACTION, SFC_STEP, and SFC_STOP tag types.
- Addition of 38400 as a supported serial port baud rate.
- Addition of structured text instructions.
- Addition of EOT, SFR, and SFP instructions to relay ladder and structured text.
- Addition of sequential function chart components.
- Addition of an appendix that lists the valid CommMethod and ConfigMethod values for the supported I/O modules.

Beginning with version 2.2, multi-line rung comments (with hard returns) are no longer exported as one long string (in double-quotes). Instead, each line of a multi-line rung comment is on a separate line in the .L5K file with double-quotes around each line. When imported, the multiple quoted strings are concatenated to form the rung comment. This improves the readability of the .L5K text file using the existing multiple-string capability of the rung comment syntax. Older formats still work on import.

Import/Export Version 2.1 RSLogix 5000 Version 10

Version 2.1 (major revision 2, minor revision 1) of the import/export feature that is included with RSLogix 5000 programming software, version 10.0 included these major enhancements.

- Removal of the characters **/A** when specifying a controller type.
- Addition of the SecurityCode attribute to the Controller object.
- Enhancements to the Message tag structure (see page 225).
- The Program object now includes a Mode attribute.
- Correction to valid values for Watchdog and Rate attributes of the Task object.
- Addition of MaxStationAddress and TokenHoldFactor attributes to the Config DF1 object.
- Addition of new instructions: SIZE, SWPB, LOWER, and UPPER.
- The NumberOfAppendChars of the Config ASCII object is no longer exported. If you have an import/export file with any of these attributes, the file will correctly import into the software. This attributes will be removed when you later export the file.

Changes to Support MESSAGE Tag Enhancements

Version 2.1 (major revision 2, minor revision 1) of the import/export feature that is included with RSLogix 5000 programming software, version 10.0 made significant changes to the MESSAGE tag. For reference, the following table shows the MESSAGE tag structure of the previous import/export release.

MESSAGE Tag Structure (Version 2.0)

Attribute	Description
`Description`	Provide information about the tag. Specify `Description := "text"`
`Comment`	Provide information about a tag component. Specify `Comment<specifier> := "text"` Where the *specifier* is: *.bitnumber* for a bit in the tag [*element*] for an array element of the tag *.membername* for a structure member of the tag
`MessageType`	Enter Block Transfer Read, Block Transfer Write, CIP Data Table Read, CIP Data Table Write, CIP Generic, PLC2 Unprotected Read, PLC2 Unprotected Write, PLC3 Typed Read, PLC3 Typed Write, PLC3 Word Range Read, PLC3 Word Range Write, PLC5 Typed Read, PLC5 Typed Write, PLC5 Word Range Read, PLC5 Word Range Write, SLC Typed Read, or SLC Typed Write. Specify `MessageType := text`
`RequestedLength`	Specify the number of elements in the message instruction (0...32,767). Specify `RequestedLength := value`
`ConnectionPath`	Specify the connection path to the other device. Specify `ConnectionPath := string`
`DF1DHFlag`	If the communication method uses DH+, enter 1. If the communication method does not use DH+, enter 0. Specify `DF1DHFlag := value`
`LocalTag`	Specify the tag name of the element in the local device. Specify `LocalTag := text`
`RemoteElement`	Specify the tag name of the element in the remote device. Specify `RemoteElement := value`
`DHPlusSourceLink`	If the communication method uses DH+, specify the source link (0...65,535). Specify `DHPlusSourceLink := value`
`DHPlusDestinationLink`	If the communication method uses DH+, specify the destination link (0...65,535). Specify `DHPlusDestinationLink := value`
`DHPlusDestinationNode`	If the communication method uses DH+, specify the destination node number (0...63 octal). Specify `DHPlusDestinationNode := value`
`DHPlusChannel`	If the communication method uses DH+, specify the DH+ channel. Enter either A or B. Specify `DHPlusChannel := letter`
`CacheConnections`	If the message is to cache connections, enter TRUE. If the message is not to cache connections, enter FALSE. Specify `CacheConnections := text`
`ServiceCode`	If the message type is CIP Generic, specify the service code (0...255 hexadecimal). Specify `ServiceCode := #16value`
`ObjectType`	If the message type is CIP Generic, specify the object type (0...65,535 hexadecimal). Specify `ObjectType := 16#value`

Attribute	Description
`TargetObject`	If the message type is CIP Generic, specify the target object (0...65,535 decimal). Specify `TargetObject := value`
`AttributeNumber`	If the message type is CIP Generic, specify the attribute number (0...65,535 hexadecimal). Specify `AttributeNumber := 16#value`
`DestinationTag`	Specify the tag name of the destination element. Specify `DestinationTag := text`

Import/Export Version 2.0 RSLogix 5000 Version 9

Version 2.0 (major revision 2, minor revision 0) of the import/export feature that is included with RSLogix 5000 programming software, version 9.0 included these major enhancements.

- The AXIS tag was replaced with AXIS_CONSUMED, AXIS_SERVO, AXIS_SERVO_DRIVE, and AXIS_VIRTUAL tags.
- For any attribute that you can specify a "not applicable" state, you must enter <NA>, rather than just NA.
- This revision of the manual includes a description and example of the STRING data type.

IMPORTANT Version 9 of RSLogix 5000 programming software only supports ControlLogix processors.

Motion Changes to Support the SERCOS Protocol

Version 2.0 (major revision 2, minor revision 0) of the import/export feature that is included with RSLogix 5000 programming software, version 9.0 made significant changes to motion-related tags to support the SERCOS protocol.

- CoarseUpdatePeriod and AutoTagUpdate parameters were added to the MOTION_GROUP tag to support SERCOS. For reference, the previous structure is described below (page 229).

- Earlier versions of the import/export feature supported one AXIS tag. To support SERCOS, the import/export feature replaced AXIS with four axis tags: AXIS_CONSUMED, AXIS_SERVO, AXIS_SERVO_DRIVE, and AXIS_VIRTUAL. The previous AXIS tag is incorporated into these new tags, but no longer exists as its own tag. For reference, the AXIS structure is described below (page 229).

  If you have a version 8.0 import/export file with AXIS tags that you import into version 9.0 software (after changing the import/export version line to 2.0), the AXIS tags convert to:

If the AXIS type is	It Converts to
Unused	AXIS_SERVO
Position only	AXIS_SERVO
Servo	AXIS_SERVO
Consumed	AXIS_CONSUMED
Virtual	AXIS_VIRTUAL

MOTION_GROUP Tag Structure (Version 1.1)

Attribute	Description
`Description`	Provide information about the tag. Specify `Description := "text"`
`Comment`	Provide information about a tag component. Specify `Comment<specifier> := "text"` Where the *specifier* is: *.bitnumber* for a bit in the tag [*element*] for an array element of the tag *.membername* for a structure member of the tag
`GroupType`	Specify the type of motion group, such as Independent. Specify `GroupType := text`
`CoarseUpdateMultiplier`	Specify the coarse update rate (5-320ms). Specify `CoarseUpdateMultiplier := value`
`ServoUpdatePeriod`	Specify the servo update period in milliseconds (any positive number) Specify `ServoUpdatePeriod := value`
`PhaseShift`	Specify the phase shift (0-65,535). Specify `PhaseShift := value`
`GeneralFaultType`	Specify whether an error generates a major fault or a non-major fault. Enter Major Fault or Non Major Fault. Specify `GeneralFaultType := text`

AXIS Tag Structure (Version 1.1)

Attribute	Description
`Description`	Provide information about the tag. Specify `Description := "text"`
`Comment`	Provide information about a tag component. Specify `Comment<specifier> := "text"` Where the *specifier* is: *.bitnumber* for a bit in the tag [*element*] for an array element of the tag *.membername* for a structure member of the tag
`MotionGroup`	Enter the name of the associated motion group, or enter NA. Specify `MotionGroup := text`
`MotionModule`	Enter the name of the associated motion module, or enter NA. Specify `MotionModule := text`
`AxisState`	Enter Axis-Ready, Direct Drive Control, Servo Control, Axis Faulted, or Axis Shutdown. Specify `AxisState := text`
`PositionUnits`	Specify the type of units. Specify `PositionUnits := text`
`TimeUnits`	Enter Seconds or Minutes. Specify `TimeUnits := text`

Attribute	Description
`InstructionSpeedUnits`	Enter Percentage or Engineering Units. Specify `InstructionSpeedUnits := text`
`InstructionAccelDecelUnits`	Enter Percentage or Engineering Units. Specify `InstructionAccelDecelUnits := text`
`InstructionMoveProfile`	Enter Trapezoidal or S-Curve. Specify `InstructionMoveProfile := text`
`InstructionJogProfile`	Specify Trapezoidal or S-Curve. Specify `InstructionJogProfile := text`
`ConversionConstant`	Specify the conversion constant. Enter a real number from 1.0...1.0e^{9}. Specify `ConversionConstant := value`
`HomeMode`	Enter Passive or Active. Specify `HomeMode := text`
`HomeSequenceType`	Enter Immediate Home, Home To Switch, Home To Marker Only, or Home To Switch With Marker. Specify `HomeSequenceType := text`
`HomePosition`	Specify the home position (any positive number). Specify `HomePosition := value`
`HomeSpeed`	Specify the home speed (any positive number). Specify `HomeSpeed := value`
`HomeReturnSpeed`	Specify the home return speed (any positive number). Specify `HomeReturnSpeed := value`
`MaximumSpeed`	Specify the maximum speed (any positive number). Specify `MaximumSpeed := value`
`MaximumAcceleration`	Specify the maximum acceleration (any positive number). Specify `MaximumAcceleration := value`
`MaximumDeceleration`	Specify the maximum deceleration (any positive number). Specify `MaximumDeceleration := value`
`ProgrammedStopMode`	Enter Fast Stop, Fast Shutdown, or Hard Shutdown. Specify `ProgrammedStopMode := text`
`AverageVelocityTimebase`	Specify the average velocity timebase (any positive number). Specify `AverageVelocityTimebase := value`
`ServoStatusUpdateBits`	Specify the servo status update bits. Enter a hexadecimal number. Specify `ServoStatusUpdateBits := 16#value`
`MotionConfigurationBits`	Specify the motion configuration bits. Enter a hexadecimal number. Specify `MotionConfigurationBits := 16#value`
`AxisType`	Enter Unused, Position Only, Servo, Consumed, or Virtual. Specify `AxisType := text`
`PositionUnwind`	Specify the unwind position (0-65,535). Specify `PositionUnwind := value`
`MaximumPositiveTravel`	Specify the maximum positive travel (any positive number). Specify `MaximumPositiveTravel := value`
`MaximumNegativeTravel`	Specify the maximum negative travel (any positive number). Specify `MaximumNegativeTravel := value`
`PositionErrorTolerance`	Specify the position error tolerance (any positive number). Specify `PositionErrorTolerance := value`

Attribute	Description
PositionLockTolerance	Specify the position local tolerance (any positive number). Specify `PositionLockTolerance := value`
PositionProportionalGain	Specify position proportional gain (any positive number). Specify `PositionProportionalGain := value`
PositionIntegralGain	Specify the position integral gain (any positive number). Specify `PositionIntegralGain := value`
VelocityFeedforwardGain	Specify the velocity feedforward gain (any positive number). Specify `VelocityFeedforwardGain := value`
AcclerationFeedforwardGain	Specify the acceleration feedforward gain (any positive number). Specify `AccelerationFeedforwardGain := value`
VelocityProportionalGain	Specify the velocity proportional gain (any positive number). Specify `VelocityProportionalGain := value`
VelocityIntegralGain	Specify velocity integral gain (any positive number). Specify `VelocityIntegralGain := value`
OutputFilterBandwidth	Specify output filter bandwidth (any positive number). Specify `OutputFilterBandwidth := value`
OutputScaling	Specify the output scaling (any positive number). Specify `OutputScaling := value`
OutputLimit	Specify the output limit (any positive number). Specify `OutputLimit := value`
OutputOffset	Specify output offset (any positive number). Specify `OutputOffset := value`
FrictionCompensation	Specify friction compensation (any positive number). Specify `FrictionCompensation := value`
SoftOvertravelFaultAction	Enter Shutdown, Disable Drive, Stop Motion, or Status Only. Specify `SoftOvertravelFaultAction := text`
PositionErrorFaultAction	Enter Shutdown, Disable Drive, Stop Motion, or Status Only. Specify `PositionErrorFaultAction := text`
EncoderLossFaultAction	Enter Shutdown, Disable Drive, Stop Motion, or Status Only. Specify `EncoderLossFaultAction := text`
EncoderNoiseFaultAction	Enter Shutdown, Disable Drive, Stop Motion, or Status Only. Specify `EncoderNoiseFaultAction := text`
DriveFaultAction	Enter Shutdown, Disable Drive, Stop Motion, or Status Only. Specify `DriveFaultAction := text`
ServoConfigurationBits	Specify the servo configuration bits. Enter a hexadecimal number. Specify `ServoConfigurationBits := 16#value`
MotorEncoderTestIncrement	Specify the motor encoder test increment (any positive number). Specify `MotorEncoderTestIncrement := value`
TuningTravelLimit	Specify the tuning travel limit (any positive number). Specify `TuningTravelLimit := value`
TuningSpeed	Specify the tuning speed (any positive number). Specify `TuningSpeed := value`

Attribute	Description
`DampingFactor`	Specify the damping factor (any positive number). Specify `DampingFactor := value`
`PositionServoBandwidth`	Specify position servo bandwidth (any positive number). Specify `PositionServoBandwidth := value`
`TuningConfigurationBits`	Specify the tuning configuration bits. Enter a hexadecimal number. Specify `TuningConfigurationBits := 16#value`

Import/Export Version 1.1 RSLogix 5000 Version 8

Version 1.1 (major revision 1, minor revision 1) of the import/export feature that is included with RSLogix 5000 programming software, version 8.0 included these major enhancements.

- Addition of function block instructions and routines.
- Addition of ASCII instructions.
- Verification of all instruction attributes and parameters.

A

ACTION_LIST block 143
ADD_ON_INSTRUCTION_DEFINITION
attributes 57
component 55
example 58
guidelines 58
parameters 60
source protected 62, 99
add-on instruction 55, 58
AddOnInstructionDefinition
component 206
ALARM_ANALOG
ALMMSG 72
attributes 69
ALARM_DIGITAL
ALMMSG 74
attributes 72
aliases 67
ALMMSG
ALARM_ANALOG 72
ALARM_DIGITAL 74
record 193
array specifications 67
ATTACHMENT
entering 124, 153
example 124, 153
guidelines 124, 153
attachments 124, 153
attributes
ADD_ON_INSTRUCTION_DEFINITION 57
ALARM_ANALOG 69
ALARM_DIGITAL 72
AXIS_CONSUMED TAG 75
AXIS_SERVO TAG 75
AXIS_SERVO_DRIVE TAG 75
AXIS_VIRTUAL TAG 75
CONTROLLER 28
COORDINATE_SYSTEM TAG 84
DATATYPE 41
EquipmentPhase PROGRAM 97
FBD_ROUTINE 112
LOCAL_TAGS 61
MESSAGE TAG 86
MODULE 46
MOTION_GROUP TAG 87
PEN declaration 176
PROGRAM 96
attributed (continued)
ROUTINE 101
safety CONTROLLER 29
safety MODULE 48
safety TAG 88
SFC_ROUTINE 132
ST_ROUTINE 155
TAG 67
TASK 168, 180
TREND 172
AXIS_CONSUMED TAG 75
AXIS_SERVO TAG 75
AXIS_SERVO_DRIVE TAG 75
AXIS_VIRTUAL TAG 75

B

backward compatibility 222
BLOCK
component 121
guidelines 122
block 143
BRANCH
entering 149
example 150
LEG block 150
branches 103, 149

C

COMMENT
record 191
comments 187
CSV format 187
internal file 24
rung logic 104
structured text logic 157
TAG 90
TXT Unicode format 187
complete
ADD_ON_INSTRUCTION_DEFINITION 55, 58
branches 103
comments 24
components 22
CONFIG 179
connection list 49
conventions 23
DATATYPE 41
display style 24
FBD_ROUTINE 111

complete (continued)
function block logic 112
MODULE 45
PROGRAM 95
ROUTINE 101
rung logic 102
sequential function chart logic 131
SFC_ROUTINE 131
ST_ROUTINE 155
structured text logic 156
TAG 65
TASK 167
TREND 171
complete import/export 14, 15
components
ADD_ON_INSTRUCTION_DEFINITION 55
AddOnInstructionDefinition 206
basic format 22
BLOCK 121
CONFIG 179
DATATYPE 41
DataType 202
descriptions 24
display style 24
FBD_ROUTINE 111
ICON 119
IREF 117
MODULE 45
Module 205
OCON 119
OREF 117
PROGRAM 95
Program 210
ROUTINE 101
SFC_ROUTINE 131
ST_ROUTINE 155
TAG 65
Tag 209
TASK 167
TREND 171
Trend 214
WIRE 120
CONDITION block 146
CONFIG
component 179
examples 184
connection list 49
CONTROLLER
attributes 28
example 31
guidelines 30
safety attributes 29
safety example 35
controller objects 179
conventions 23
COORDINATE_SYSTEM TAG 84
CSV format 16, 217
examples 194

D

DATATYPE
attributes 41
component 41
example 44
guidelines 44
DataType
component 202
descriptions 24
dimensions 67
directed links 151
DIRECTED_LINK
entering 151
example 151
guidelines 151
display style 24

E

ENCODED_DATA 62, 99
entering
attachments 124, 153
branches 149
directed links 151
steps 141
stops 148
subroutine calls 147
text boxes 152
transitions 145
EquipmentPhase PROGRAM 97
examples
ADD_ON_INSTRUCTION_DEFINITION 58
ATTACHMENT 124, 153
BRANCH 150
CONFIG 184
CONTROLLER 31
CSV files 194

examples (continued)
DATATYPE 44
DIRECTED_LINK 151
EquipmentPhase PROGRAM 98
function block logic 114
ICON 120
IREF 118
LOGIC 116, 139, 158
MODULE 51
OCON 120
online fsequential function chart logic 139
online function block logic 116
online structured text logic 158
OREF 118
PROGRAM 98
ROUTINE 103
rung logic 104
safety CONTROLLER 35
safety partner MODULE 53
safety PROGRAM 99
safety TASK 169
SBR_RET 148
sequential function chart logic 133
SFC_ROUTINE 133
SHEET 114
ST_ROUTINE 157
STEP 145
STOP 149
TAG 90, 91
TASK 169
TEXT_BOX 123, 152
TRANSITION 147
TREND 178
TXT Unicode files 196
WIRE 121
Excel 217
exporting
complete project 14
CSV format 16
file structure 202
L5K format 14
L5X format 19
ladder rungs 19
partial project 16, 19
project 14
tags 16, 19
trends 19, 177
types 13

F

FBD_ROUTINE
attributes 112
BLOCK logic 121
component 111
example 114
ICON logic 119
IREF logic 117
LOGIC block 116
OCON logic 119
OREF logic 117
SHEET logic 113
WIRE logic 120
format
CSV 187
L5X 177, 199
TXT Unicode 187
function block logic 112
entering attachments 124
online edits 116
structure 111

G

guidelines
ADD_ON_INSTRUCTION_DEFINITION 58
ATTACHMENT 124, 153
BLOCK logic 122
CONTROLLER 30
DATATYPE 44
DIRECTED_LINK 151
ICON logic 119
IREF logic 118
MODULE 51
OCON 119
OREF
logic 118
PROGRAM 97
rung logic 102
SHEET logic 113
TAG 90
TASK 168
TEXT_BOX 123, 152
TREND 177
WIRE logic 121

H

history, import/export feature 221

I

ICON
component 119
example 120
guidelines 119
importing
complete project 15
CSV format 17
file structure 202
L5K format 15
L5X format 20
ladder rungs 20
partial project 17, 20
project 15
tags 17, 20
trends 20
types 13
initial values 89
instructions 104, 125, 159
internal file comments 24, 187
IREF
component 117
example 118
guidelines 118

L

L5X format 19, 177, 199
ladder logic 101
LEG block 150
LIMIT_HIGH block 143
LIMIT_LOW block 143
LOCAL_TAGS
attributes 61
logic 102, 112, 131, 156
LOGIC block 116, 139, 158

M

MESSAGE TAG 86
Microsoft Excel 217
MODULE
attributes 46
component 45
connection list 49
example 51
guidelines 51
safety attributes 48
safety partner example 53
Module
component 205
MOTION_GROUP TAG 87

N

neutral text 104, 125, 159

O

objects 179
OCON
component 119
example 120
guidelines 119
online function block 116
online sequential function chart 139
online structured text 158
OREF
component 117
example 118
guidelines 118
overview 202

P

parameters
ADD_ON_INSTRUCTION_DEFINITION 60
partial
AddOnInstructionDefinition 206
ALMMSG record 193
COMMENT record 191
comments 187
CSV format 187
DataType 202
L5X format 199
Module 205
Program 210
RCOMMENT 187
remark 187
structure 202
TAG 187
Tag 209
TAG record 188
TREND 177
Trend 214
TXT Unicode format 187
using Excel 217
partial import/export 16, 17, 19, 20
PEN declaration 176
phase manager program 97

PRESET block 142
PROGRAM
attributes 96
component 95
EquipmentPhase attributes 97
EquipmentPhase example 98
example 98
guidelines 97
Program
component 210
projects 14, 15

R

RCOMMENT
partial 187
record 191
remark 187
ROUTINE
attributes 101
component 101
example 103
rung logic 102
rungs 19, 20

S

safety controller
CONTROLLER attributes 29
example 35
MODULE attributes 48
MODULE example 53
PROGRAM example 99
TAG 88
TAG example 91
TASK example 169
SBR_RET
entering 147
example 148
sequential function chart logic
ACTION_LIST block 143
CONDITION block 146
entering attachments 153
entering branches 149
entering directed links 151
entering steps 141
entering stops 148
entering subroutine calls 147
entering text boxes 152
entering transitions 145
sequential function chart logic (continued)
example 133
LEG block 150
LIMIT_HIGH block 143
LIMIT_LOW block 143
online edits 139
PRESET block 142
structure 131
SFC_ROUTINE
attributes 132
component 131
example 133
LOGIC block 139
SHEET
example 114
source protected 62, 99
ST_ROUTINE
attributes 155
component 155
example 157
LOGIC block 158
STEP
ACTION_LIST 143
entering 141
example 145
LIMIT_HIGH block 143
LIMIT_LOW block 143
PRESET block 142
STOP
entering 148
example 149
structure 202
structured text logic 156
entering 159
online edits 158
routine 157
structure 155
subroutine calls 147

T

TAG
aliases 67
array specifications 67
attributes 67
AXIS_CONSUMED 75
AXIS_SERVO 75
AXIS_SERVO_DRIVE 75
AXIS_VIRTUAL 75
component 65

TAG (continued)
component comments 90
COORDINATE_SYSTEM 84
example 90
guidelines 90
initial values 89
MESSAGE 86
MOTION_GROUP 87
partial 187
record 188
safety 88
Tag
component 209
tags 16, 17, 19, 20
TASK
attributes 168, 180
component 167
example 169
guidelines 168
text boxes 152
text file 14
TEXT_BOX
entering 152
example 123, 152
guidelines 123, 152
TEXTBOX record 191
TRANSITION
CONDITION block 146
entering 145
example 147
TREND
attributes 172
component 171
example 178
guidelines 177
partial 177
PEN declaration 176
Trend
component 214
TXT Unicode format
examples 196

U

Unicode TXT format 187

W

WIRE
component 120
example 121
guidelines 121

How Are We Doing?

Your comments on our technical publications will help us serve you better in the future. Thank you for taking the time to provide us feedback.

You can complete this form and mail it back to us, visit us online at www.ab.com/manuals, or email us at RADocumentComments@ra.rockwell.com

Pub. Title/Type Logix5000 Controllers Import/Export Reference Manual

Cat. No. Logix-based controllers Pub. No. 1756-RM084L-EN-P Pub. Date January 2007 Part No. 953030-21

Please complete the sections below. Where applicable, rank the feature (1=needs improvement, 2=satisfactory, and 3=outstanding).

Overall Usefulness 1 2 3	How can we make this publication more useful for you?	
Completeness (all necessary information is provided) 1 2 3	Can we add more information to help you? ☐ procedure/step ☐ illustration ☐ feature ☐ example ☐ guideline ☐ other ☐ explanation ☐ definition	
Technical Accuracy (all provided information is correct) 1 2 3	Can we be more accurate? ☐ text ☐ illustration	
Clarity (all provided information is easy to understand) 1 2 3	How can we make things clearer?	
Other Comments	You can add additional comments on the back of this form.	

Your Name _____ Location/Phone _____

Your Title/Function _____ Would you like us to contact you regarding your comments?

___No, there is no need to contact me

___Yes, please call me

___Yes, please email me at _____

___Yes, please contact me via _____

Return this form to: Allen-Bradley Marketing Communications, 1 Allen-Bradley Dr., Mayfield Hts., OH 44124-9705

Phone: 440-646-3176 Fax: 440-646-3525 Email: RADocumentComments@ra.rockwell.com

PLEASE FASTEN HERE (DO NOT STAPLE)

Other Comments

PLEASE FOLD HERE

PLEASE REMOVE

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST-CLASS MAIL PERMIT NO. 18235 CLEVELAND OH

POSTAGE WILL BE PAID BY THE ADDRESSEE

Allen-Bradley
RELIANCE ELECTRIC
DODGE
ROCKWELL SOFTWARE
Rockwell Automation

1 ALLEN-BRADLEY DR
MAYFIELD HEIGHTS OH 44124-9705

Notes:

Notes:

Rockwell Automation Support

Rockwell Automation provides technical information on the Web to assist you in using its products. At http://support.rockwellautomation.com, you can find technical manuals, a knowledge base of FAQs, technical and application notes, sample code and links to software service packs, and a MySupport feature that you can customize to make the best use of these tools.

For an additional level of technical phone support for installation, configuration, and troubleshooting, we offer TechConnect Support programs. For more information, contact your local distributor or Rockwell Automation representative, or visit http://support.rockwellautomation.com.

Installation Assistance

If you experience a problem with a hardware module within the first 24 hours of installation, please review the information that's contained in this manual. You can also contact a special Customer Support number for initial help in getting your module up and running.

United States	1.440.646.3223 Monday – Friday, 8am – 5pm EST
Outside United States	Please contact your local Rockwell Automation representative for any technical support issues.

New Product Satisfaction Return

Rockwell tests all of its products to ensure that they are fully operational when shipped from the manufacturing facility. However, if your product is not functioning, it may need to be returned.

United States	Contact your distributor. You must provide a Customer Support case number (see phone number above to obtain one) to your distributor in order to complete the return process.
Outside United States	Please contact your local Rockwell Automation representative for return procedure.

www.rockwellautomation.com

Power, Control and Information Solutions Headquarters

Americas: Rockwell Automation, 1201 South Second Street, Milwaukee, WI 53204-2496 USA, Tel: (1) 414.382.2000, Fax: (1) 414.382.4444
Europe/Middle East/Africa: Rockwell Automation, Vorstlaan/Boulevard du Souverain 36, 1170 Brussels, Belgium, Tel: (32) 2 663 0600, Fax: (32) 2 663 0640
Asia Pacific: Rockwell Automation, Level 14, Core F, Cyberport 3, 100 Cyberport Road, Hong Kong, Tel: (852) 2887 4788, Fax: (852) 2508 1846

Publication 1756-RM084L-EN-P - January 2007 PN 953030-21
Supersedes Publication 1756-RM084K-EN-P - May 2005
Copyright © 2007 Rockwell Automation, Inc. All rights reserved. Printed in the U.S.A.